Python
数据科学零基础一本通

上册

洪锦魁◎著

- 人工智能基础知识
- Python彩蛋
- bytes数据、编码、译码
- 用经纬度计算城市间的距离
- 用数学方法计算圆周率
- 生成式
- lambda、map( )、reduce( )
- 文件加密与解密
- 文件读取、写入与目录管理
- 程序异常处理与程序日志
- 正则表达式
- 递归式观念与碎形
- 图像、QR code、文字辨识
- GUI、动画、游戏、小计算器
- CSV和JSON文件
- 词云设计
- 股市数据读取与图表制作
- 基础统计、线性代数
- matplotlib中英文图表绘制
- Numpy、Scipy、Pandas

800多个程序实例
400多个模块
200多道实践习题

清华大学出版社

Python

数据科学零基础一本通

上册

洪锦魁◎著

清华大学出版社
北 京

内 容 简 介

这是一本专为没有编程基础的读者编写的 Python 入门书籍，全书包含 800 多个程序实例及 200 多道实践习题，一步一步详细讲解 Python 语法的基础知识，同时也将应用范围拓展至图形界面设计、影像处理、图表绘制、文字识别、词云、股市资料摘取与图表制作、线性代数、基础统计以及与数据科学相关的 Numpy、Scipy、Pandas。Python 是一门非常灵活的编程语言，本书特色在于对 Python 的基础知识与应用辅以大量实例进行讲解，读者可以通过这些程序实例事半功倍地学会 Python。

图书在版编目（CIP）数据

Python数据科学零基础一本通 / 洪锦魁著. —北京：清华大学出版社，2020.2
ISBN 978-7-302-54539-2

Ⅰ. ①P… Ⅱ. ①洪… Ⅲ. ①软件工具—程序设计 Ⅳ. ①TP311.561

中国版本图书馆 CIP 数据核字（2019）第 290377 号

责任编辑：张 敏 薛 阳
封面设计：杨玉兰
责任校对：徐俊伟
责任印制：丛怀宇

出版发行：清华大学出版社
网 址：http://www.tup.com.cn，http://www.wqbook.com
地 址：北京清华大学学研大厦 A 座 **邮 编：**100084
社 总 机：010-62770175 **邮 购：**010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质 量 反 馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 装 者：三河市铭诚印务有限公司
经 销：全国新华书店
开 本：170mm×240mm **印 张：**48.75 **字 数：**1278 千字
版 次：2020 年 4 月第 1 版 **印 次：**2020 年 4 月第 1 次印刷
定 价：129.00 元（上、下册）

产品编号：085277-01

序

多次与教育界的朋友相聚，谈到计算机语言的发展趋势时，大家一致认为 Python 是当今最重要的计算机语言。许多知名公司，例如 Google、Facebook 等皆已将 Python 列为必备计算机语言。许多人想学 Python，市面上的书也不少，但书中对 Python 语法的讲解并不完整，造成读者学习上的障碍，读者读完一本 Python 书籍，仍然看不懂专家写的 Python 程序。因此，笔者决定撰写一本用丰富、实用、有趣的实例完整且深入讲解 Python 语法的入门书籍。

Python 以简洁著名，语法非常灵活，同时拥有丰富、实用的模块。本书除了以实例解说 Python 语法，还会穿插讲解各种模块，以帮助读者更灵活地掌握 Python。此外，笔者也尝试在书中穿插基本的科学、数学、统计与人工智能的基础知识，帮助读者为进一步的学习打下扎实的基础。

本书包含 800 多个程序实例，搭配 400 多个模块，并辅以 200 多道实践习题，细致讲解 Python 语法。本书也会说明下列知识与应用：

- 人工智能基础知识；
- Python 彩蛋；
- 从 bytes 数据、编码（encode）、译码（decode）说起，到精通列表（list）、元组（tuple）、字典（dict）、集合（set）；
- 从小型列表、元组、字典到大型数据资料的建立；
- 生成式（generator）建立 Python 数据结构；
- 在坐标轴内计算任意两点之间的距离，同时解说与人工智能的关联；
- 用经纬度计算地球任意两座城市之间的距离，学习取得地球任意位置的经纬度；
- 用莱布尼茨公式、尼拉卡莎级数、蒙特卡罗模拟计算圆周率；
- 讲解基础函数观念，也深入到嵌套、closure、lambda、Decorator 等高阶应用；
- 对 map( ) 和 reduce( ) 进行完整解说，并进一步配合 lambda 解说高级应用；
- 建立类别的同时深入讲解装饰器 @property、@classmethod、@staticmethod 与类别特殊属性与方法；

- ❑ 设计与应用自己设计的模块、活用外部模块（module）；
- ❑ 赌场骗局；
- ❑ 自己设计加密与解密程序；
- ❑ Python 的输入与输出；
- ❑ 文件压缩与解压缩；
- ❑ 程序除错与异常处理；
- ❑ 文件读取与目录管理；
- ❑ 剪贴板应用；
- ❑ 正则表达式；
- ❑ 递归式观念与碎形 Fractal；
- ❑ 图像处理与文字辨识，更进一步说明计算机储存图像的方法；
- ❑ 基本与进阶 QR code 制作；
- ❑ 词云（Word Cloud）设计；
- ❑ GUI 设计：设计小计算器；
- ❑ 动画与游戏；
- ❑ matplotlib 中英文图表绘制；
- ❑ 说明 CSV 和 JSON 文件；
- ❑ 股市数据读取与图表制作；
- ❑ Python 解线性代数；
- ❑ Python 解联立方程式；
- ❑ Python 执行数据分析；
- ❑ 科学计算与数据分析 Numpy、Scipy、Pandas。

笔者编写过许多计算机领域的著作，本书将沿袭笔者以往著作的特色，程序实例丰富。相信读者通过学习本书内容，一定可以快速精通 Python。笔者虽力求完美，但是书中不足与疏漏在所难免，请不吝指正。

洪锦魁

2019.10.31

目 录

01

第 1 章

基本概念

本章摘要

1-1 认识 Python
1-2 Python 的起源
1-3 Python 语言发展史
1-4 Python 的应用范围
1-5 静态语言与动态语言
1-6 跨平台的程序语言
1-7 系统的安装与执行
1-8 Python 2 与 Python 3 不相容的验证
1-9 文件的建立、存储、执行与打开
1-10 程序注释
1-11 Python 彩蛋

1-1 认识 Python

Python 是一种**直译式**（Interpreted Language）、**面向对象**（Object Oriented Language）的程序语言，它拥有完整的函数库，可以协助用户轻松地完成许多常见的工作。

直译式语言是指，**直译器**（Interpretor）会将程序代码一句一句直接执行，不需要经过**编译**（Compile）动作，将语言先转换成**机器码**，再予以执行。目前 Python 的直译器是 CPython，这是由 C 语言编写的一个直译程序，与 Python 一样目前由 Python 基金会管理使用。

Python 也算是一种动态的高级语言，具有**垃圾回收**（garbage collection）功能。**垃圾回收**是指程序在执行时，直译程序会主动收回不再需要的动态内存空间，将内存集中管理，这种机制可以减轻程序设计师的负担，当然也就减少了程序设计师犯错的机会。

由于 Python **开放源码**（Open Source），每个人皆可免费使用或为它贡献，除了它本身有许多内建的**套件**（package）或称**模块**（module）外，许多公司也为它开发了更多的**套件**，促使它的功能可以持续扩充，因此 Python 目前已经是全球最热门的程序语言之一。

1-2 Python 的起源

Python 的最初设计者是**吉多·范罗姆苏**（Guido van Rossum），他是荷兰人，1956 年出生于荷兰哈勒姆，1982 年毕业于阿姆斯特丹大学的数学和计算机系，并获得硕士学位。



吉多·范罗姆苏在 1996 年为一本 O’Reilly 出版社作者 Mark Lutz 所著的 *Programming Python* 的序言中表示：6 年前，1989 年我想在圣诞节期间思考设计一种程序语言打发时间，当时我正在构思一个新的脚本（script）语言的解释器，它是 ABC 语言的后代，期待这个程序语言对 UNIX C 的程序语言设计师会有吸引力。基于我是蒙提派森飞行马戏团（Monty Python’s Flying Circus）的疯

狂爱好者，所以就以 Python 为这个程序命名。

在一些 Python 的文件或有些书封面喜欢用蟒蛇代表 Python，但是从吉多 • 范罗姆苏的上述序言可知，Python 灵感的来源是**马戏团名称**而非**蟒蛇**。

1999 年，他向美国国防高级研究计划局（Defense Advanced Research Projects Agency，DARPA）提出 Computer Programming for Everybody 的研发经费申请，并提出了下列 Python 的目标。

（1）这是一个简单直觉式的程序语言，可以和主要程序语言一样强大。

（2）这是开放源码（Open Source）的程序语言，每个人皆可自由使用与贡献。

（3）程序代码像英语一样容易理解与使用。

（4）可在短期间内开发一些常用功能。

现在上述目标都已经实现了，Python 已经与 C/C++、Java 一样成为程序设计师必备的程序语言，然而它却比 C/C++ 和 Java 更容易学习。

目前，Python 语言是由 Python 软件基金会（www.python.org）管理，有关新版软件下载的相关信息可以在这个基金会取得，可参考附录 A。

1-3 Python 语言发展史

在 1991 年 Python 正式诞生时，当时的操作系统平台是 Mac。尽管吉多 • 范罗姆苏坦言 Python 是构思于 ABC 语言，但是 ABC 语言并没有成功。吉多 • 范罗姆苏本人认为 ABC 语言并不是一个开放的程序语言，是其失败的主要原因。因此，在 Python 的推广中，他避开了这个错误，将 Python 推向开放式系统，因而获得了巨大的成功。

1. Python 2.0 发布

2000 年 10 月 16 日，Python 2.0 正式发布，主要是增加了**垃圾回收**的功能，同时支持 Unicode。

Unicode 是一种适合多语系的编码规则，主要是使用可变长度字节方式存储字符，以节省内存空间。例如，对于英文字母而言是使用 1 字节（byte）空间存储即可，对于含有附加符号的希腊文、拉丁文或阿拉伯文等则用 2 字节空间存储，中文则是以 3 字节空间存储，只有极少数的平面辅助文字需要 4 字节空间存储。也就是说，这种编码规则已经包含全球所有语言的字符了，所以采用这种编码方式设计程序时，其他语系的程序只要支持 Unicode 编码即可显示。例如，法国人即使使用法文版的程序，也可以正常显示中文。

2. Python 3.0 发布

2008 年 12 月 3 日，Python 3.0 正式发布。一般程序语言的发展会考虑到兼容特性，但是 Python 3 在开发时为了不受先前 2.x 版本的束缚，因此没有考虑兼容特性，所以许多早期版本开发的程序是无法在 Python 3.x 版上执行的。

不过为了解决这个问题，尽管发布了 Python 3.x 版本，后来又陆续将 3.x 版的特性移植到 Python 2.6/2.7x 版上，所以现在进入 Python 基金会网站时，可以发现有 2.7x 版和 3.7x 版的软件可以下载。

笔者经验提醒：有一些早期开发的冒险游戏软件只支持 Python 2.7x 版，目前尚未支持 Python 3.7x 版。不过相信这些软件未来也将朝向支持 Python 3.7x 版的路迈进。

Python 基金会提醒：Python 2.7x 已经被确定为最后一个 Python 2.x 的版本，目前暂定基金会对此版本的支持到 2020 年。

笔者在撰写此书时，所有程序都是以 Python 3.x 版作为主要依据的。

1-4 Python 的应用范围

尽管 Python 是一个非常适合初学者学习的程序语言，在国外有许多儿童程序语言教学也是以 Python 为工具，然而它却是一个功能强大的程序语言，下列是它的部分应用。

（1）设计动画游戏。

（2）支持图形用户接口（Graphical User Interface, GUI）开发。

（3）数据库开发与设计动态网页。

（4）科学计算与大数据分析。

（5）人工智能与机器学习。

（6）Google、Yahoo!、YouTube、NASA、Dropbox（文件分享服务）、Reddit（社交网站）在内部都大量使用 Python 作为开发工具。

（7）网络爬虫、黑客攻防。

目前，Google 搜索引擎、纽约股票交易所、NASA 航天行动的关键任务执行，都是使用 Python 语言。

1-5 静态语言与动态语言

变量（variable）是一个语言的核心，由变量的设置可以知道这个程序所要完成的工作。

有些程序语言的**变量**在使用前需要先声明它的数据类型，这样**编译程序**（compile）会在内存内预留空间给这个变量。这个变量的数据类型经过声明后，未来无法再改变它的数据类型，这类的程序语言称为**静态语言**（static language），例如，C、C++、Java 等。声明变量可以协助计算机捕捉可能的错误，同时也可以让程序执行速度更快，但是程序设计师需要花更多的时间编写程序与思考程序的规划。

有些程序语言的变量在使用前不必声明它的数据类型，这样可以用比较少的程序代码完成更多工作，增加程序设计的便利性，这类程序在执行前不必经过**编译**（compile）过程，而是使用**直译器**（interpreter）直接**直译**（interpret）与**执行**（execute），这类的程序语言称为**动态语言**（dynamic language），有时也可称这类语言是**文字码语言**（scripting language），例如，Python、Perl、Ruby。动态语言执行速度比经过编译后的静态语言执行速度慢，所以有相当长的时间动态语言只适合进行短小程序的设计，或是将它作为准备数据供静态语言处理，在这种状况下也有人将这种动态语言称为**胶水码**（glue code）。后来随着软件技术的进步，直译器执行速度越来越快，已经可以用它执行复杂的工作了。如果读者懂 Java、C、C++，将会发现，Python 相较于这些语言除了便利性，程序设计效率已经远远超过这些语言了，这也是 Python 成为目前最热门程序语言的原因。

使用 Python 语言时可以直接在提示信息下（>>>）输入程序代码执行工作（可参考 1-7 节），也可以将程序代码存储成文件然后再执行（可参考 1-9 节）。

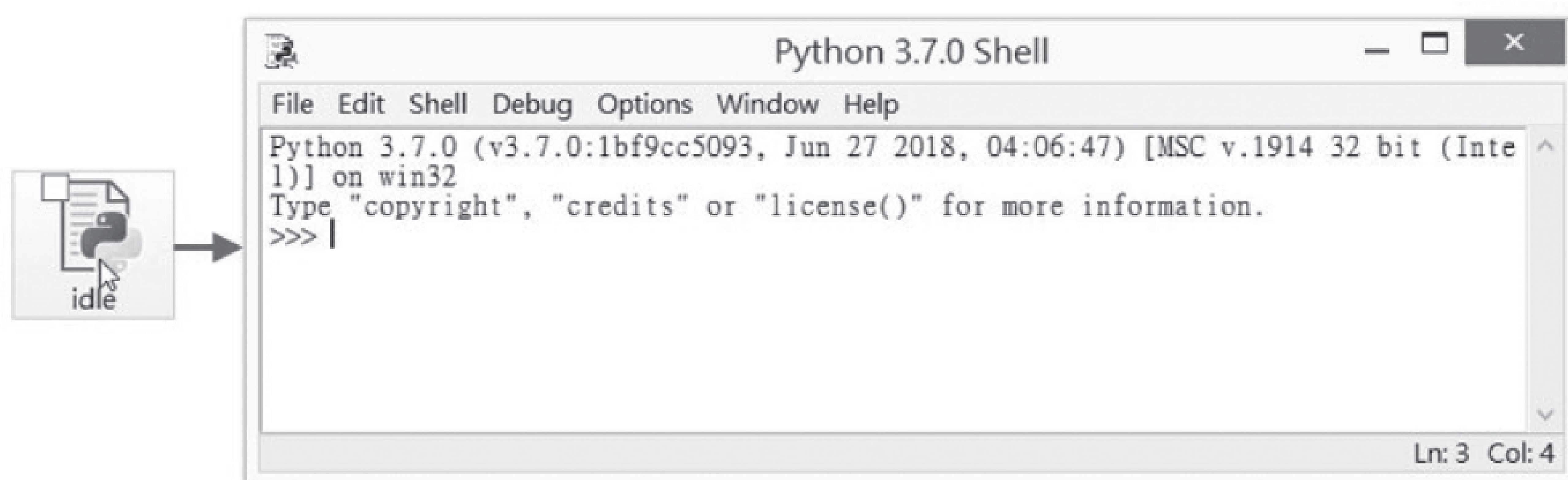
1-6 跨平台的程序语言

Python 是一种跨平台的程序语言，主要的操作系统，如 Windows、Mac OS、UNIX、Linux 等，都可以安装和使用。

跨平台的程序语言意味着，用户可以在某一个平台上使用 Python 设计一个程序，未来这个程序也可以在其他平台上顺利运行。

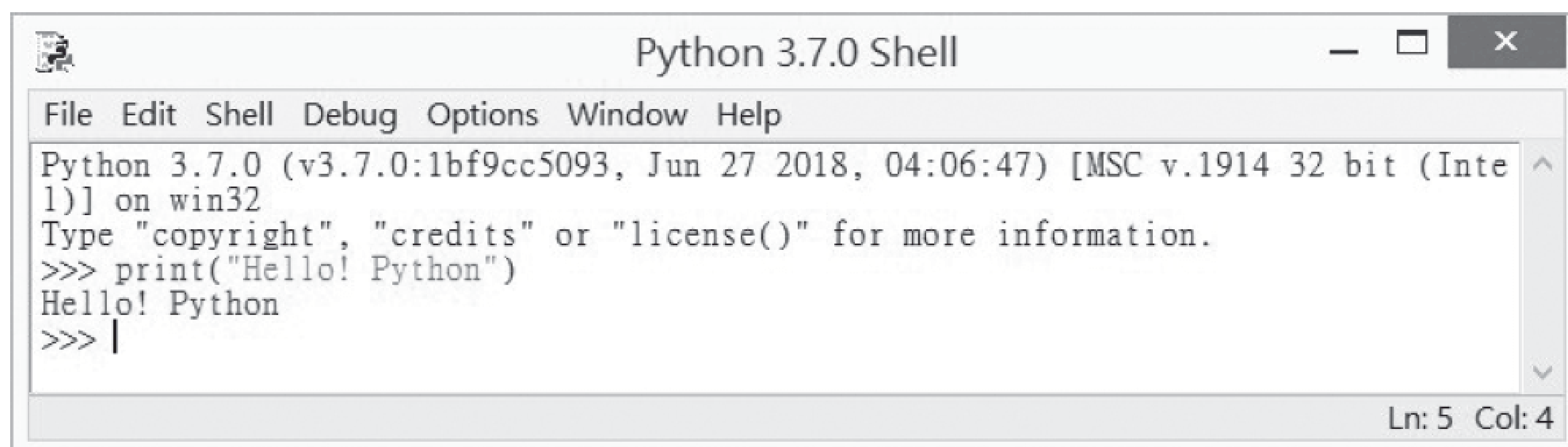
1-7 系统的安装与执行

有关安装 Python 的步骤请参考附录 A。下面将以 Python 3.7x 版为例进行说明。双击附录 A 中所建的在 Windows 桌面上的 idle 图标，将看到下列 Python Shell 窗口。



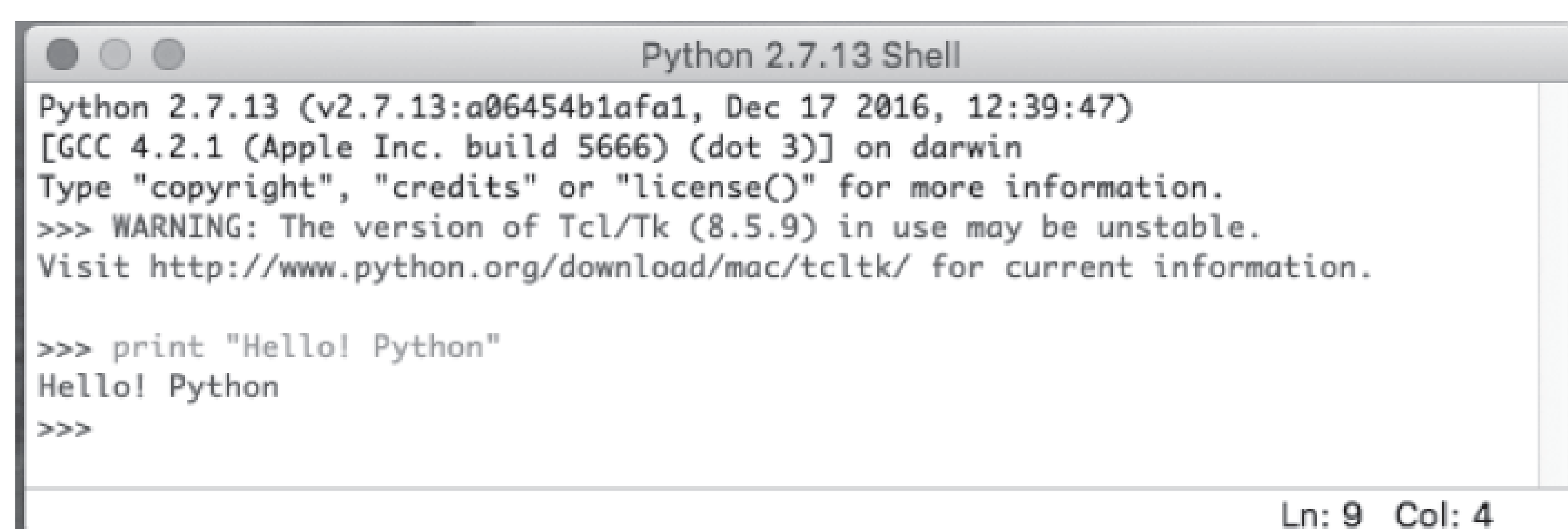
图中 >>> 符号是提示信息，可以在此输入 Python 指令。

程序实例 ch1_1.py：使用 print( ) 函数，输出字符串。

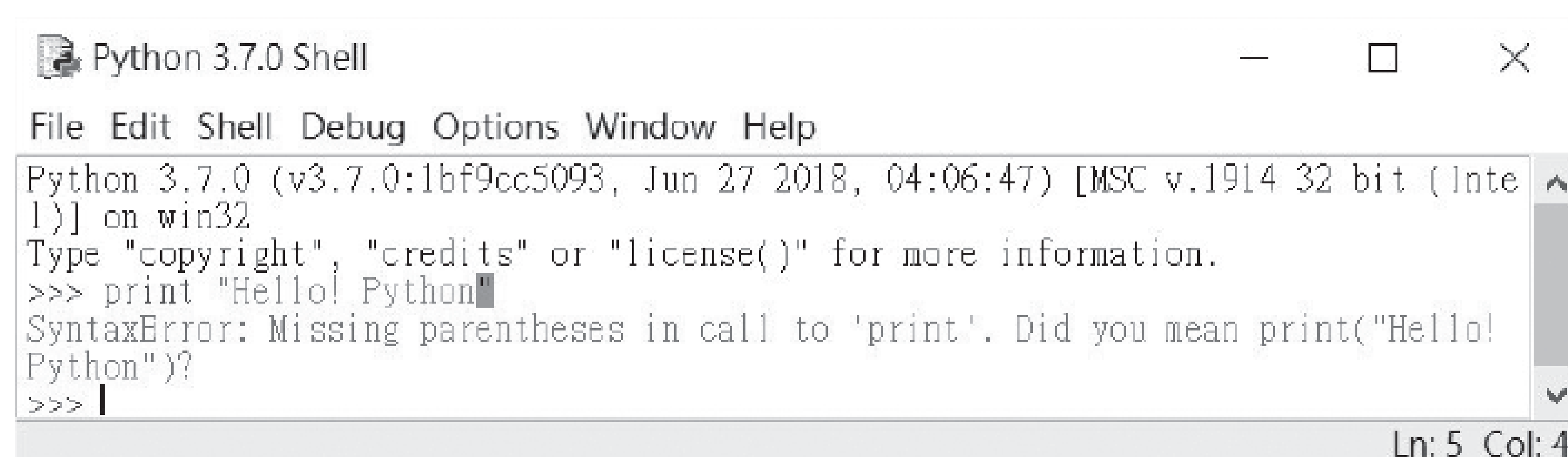


1-8 Python 2 与 Python 3 不相容的验证

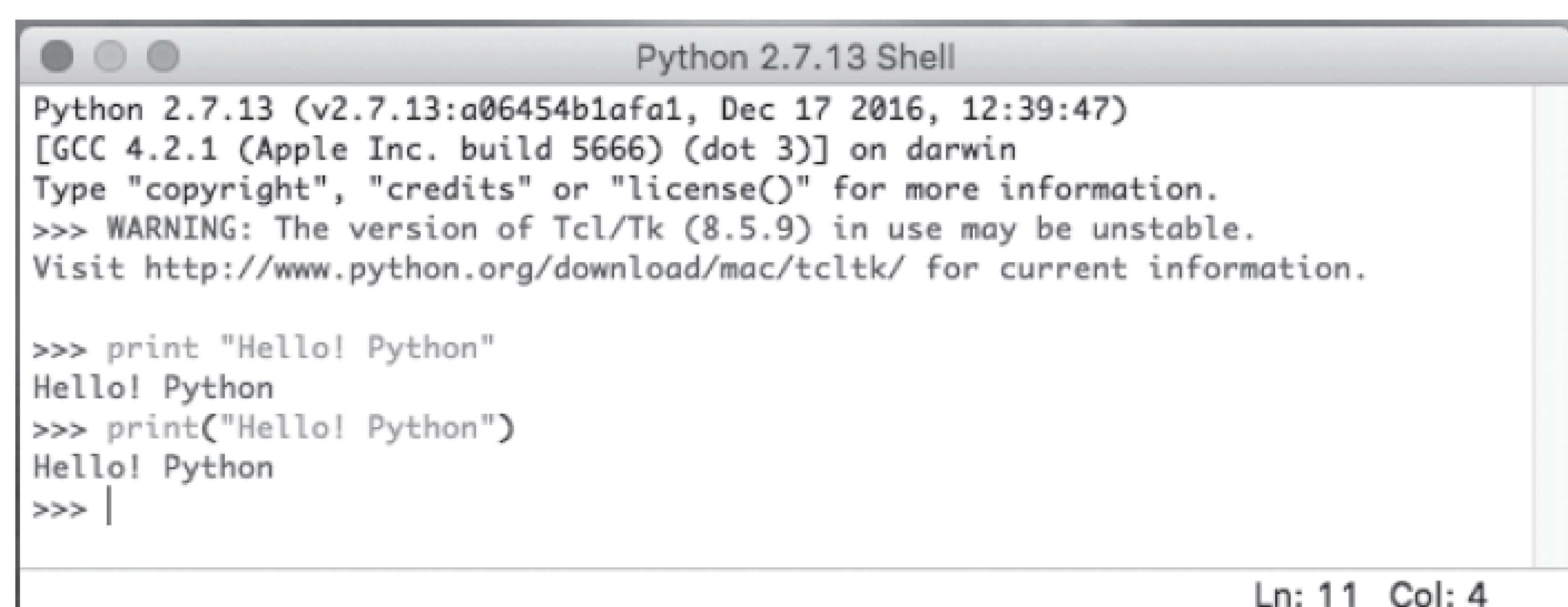
下面是早期在 Python 2 上执行输出字符串的 print 用法。



如果相同的输出方式应用在 Python 3 中将出现错误。



出现错误的原因是，在 Python 3 中 print( ) 已经是一个函数。不过在 1-3 节中也提过，Python 基金会后来陆续将 3.x 版的特性移植到 Python 2.6/2.7x 版上，所以如果在 Python 2.6/2.7x 版本上使用 print( ) 函数，将可以得到正确的输出。

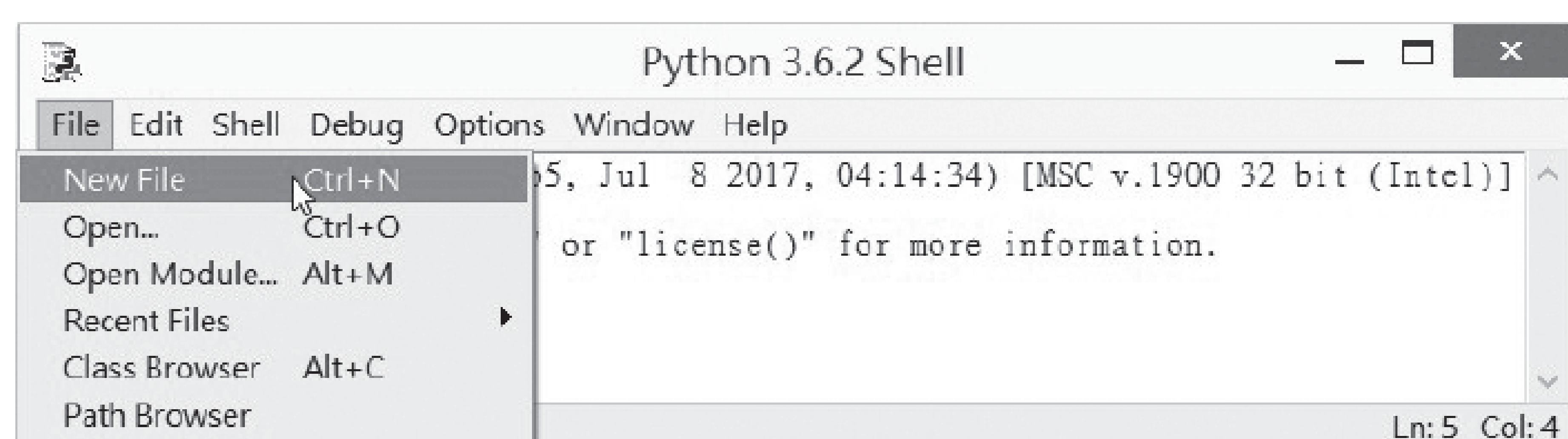


1-9 文件的建立、存储、执行与打开

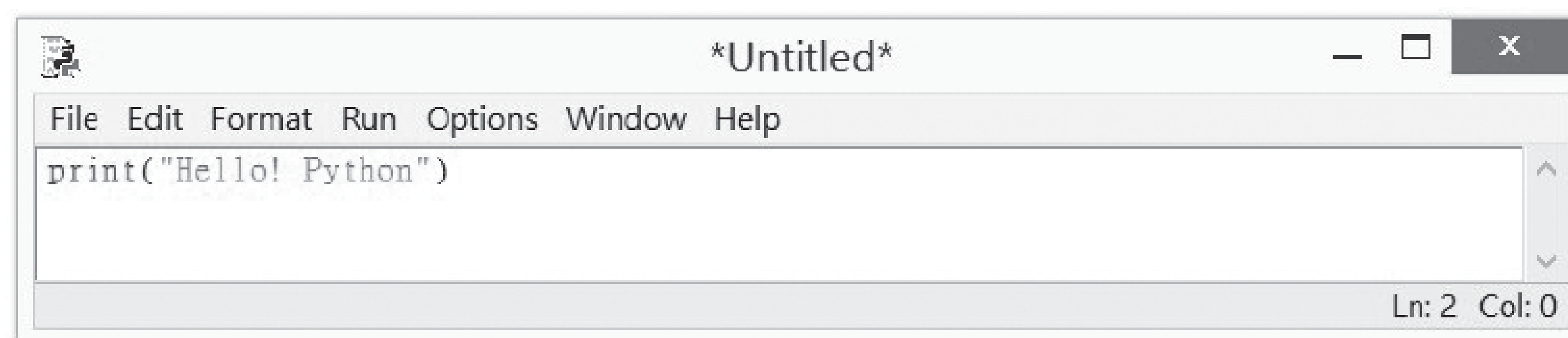
如果设计一个程序每次均要在 Python Shell 窗口环境重新输入命令的话，将是一件麻烦的事，所以程序设计时，可以将所设计的程序保存在文件内是一件重要的事。

1-9-1 文件的建立

在 Python Shell 窗口中可以执行 File → New File 命令，建立一个空白的 Python 文件。



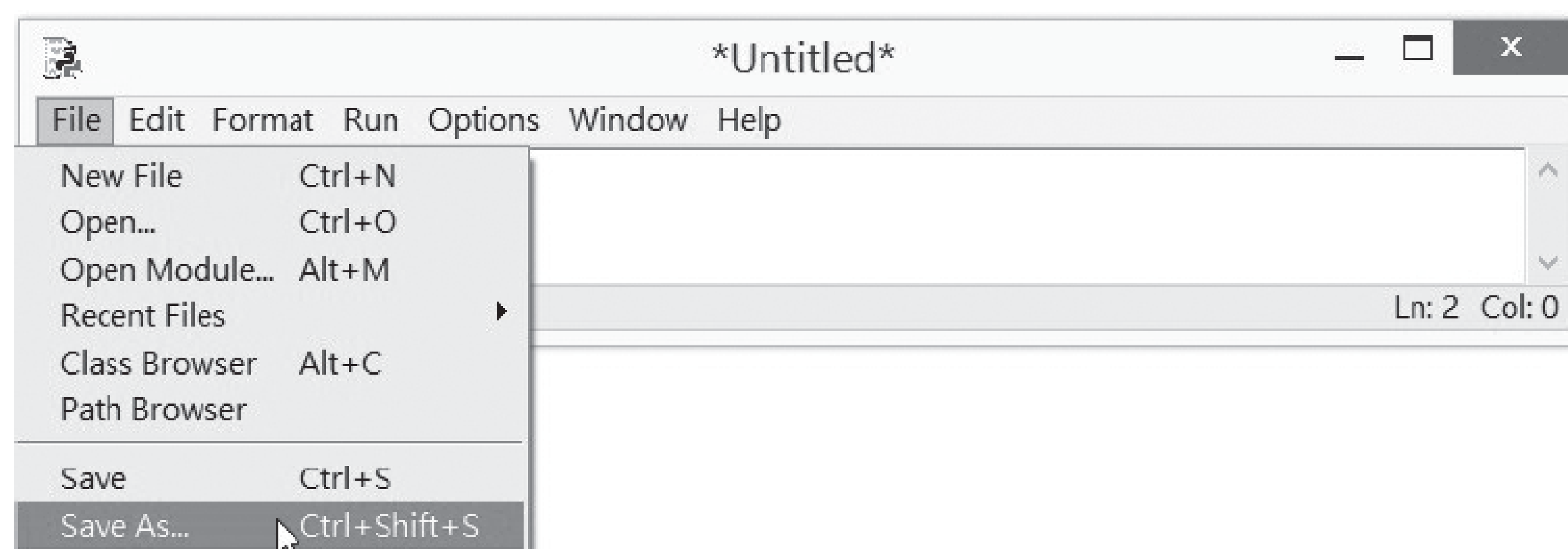
然后可以建立一个 Untitled 窗口，窗口内容是空白，下面是笔者在空白文件内输入一条命令的实例。



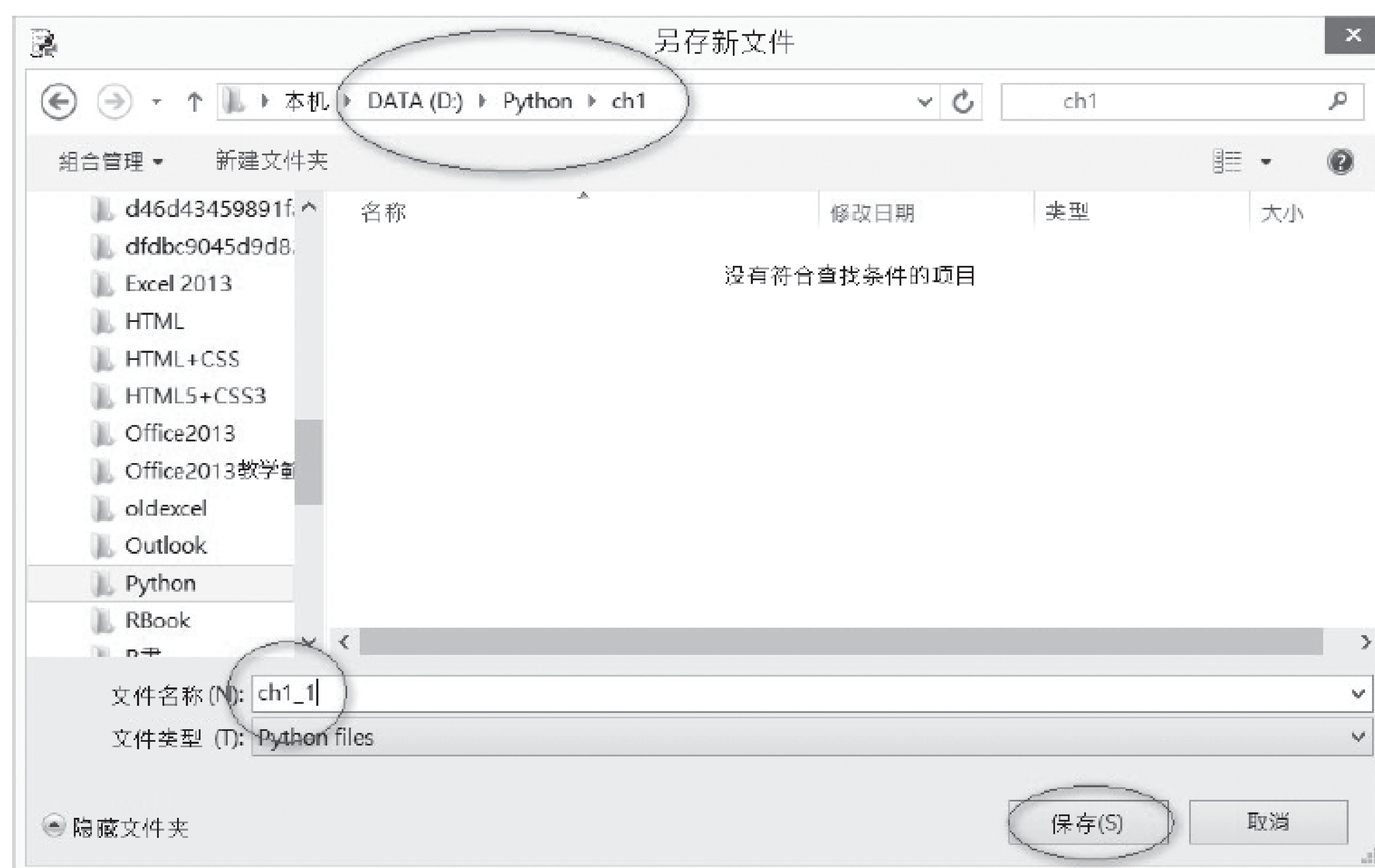
如果想要执行上述文件，需要先存储上述文件。

1-9-2　文件的存储

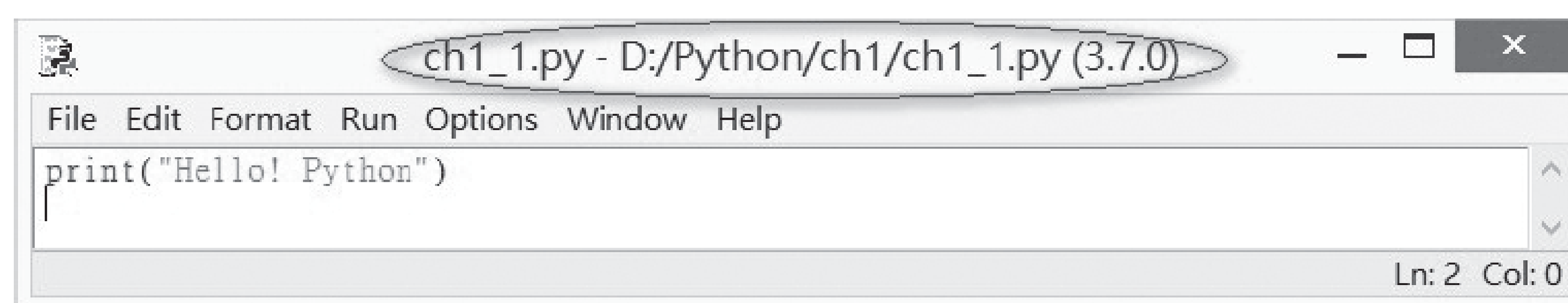
可以执行 File → Save As 命令存储文件。



然后将看到**另存新文件**对话框，此例将文件存储在 D:/Python/ch1 文件夹，文件名是 ch1_1（Python 的扩展名是 py），可以得到下列结果。



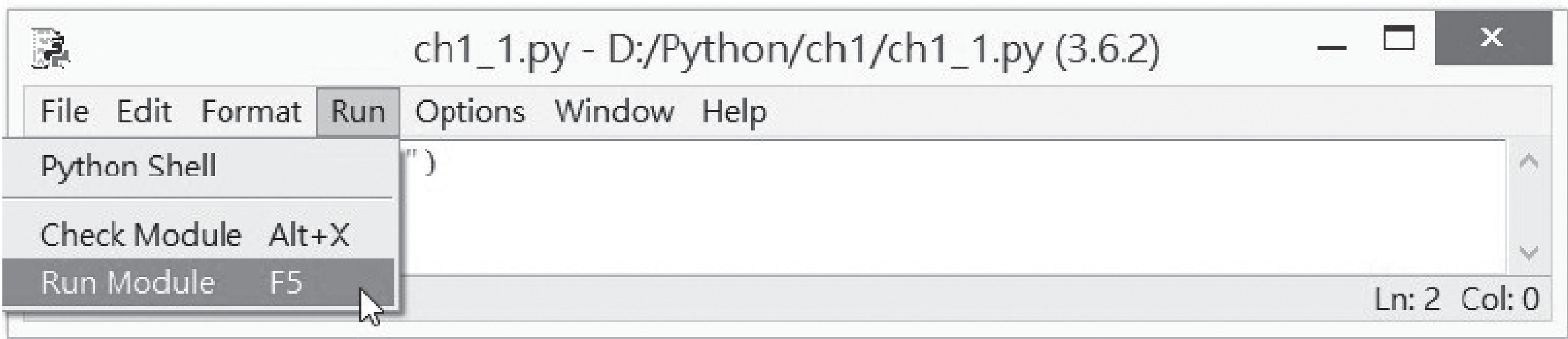
单击“保存”按钮。



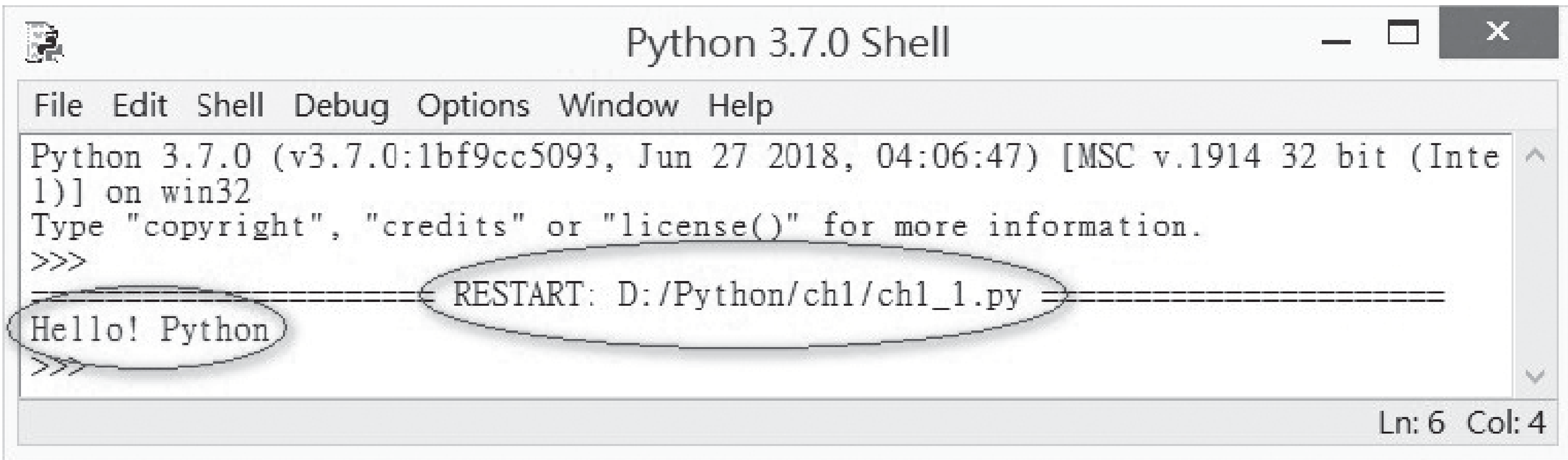
原标题 Untitled 已经改为 ch1_1.py 了。

1-9-3　文件的执行

执行 Run → Run Module 命令，就可以正式执行先前所建的 ch1_1.py 文件。



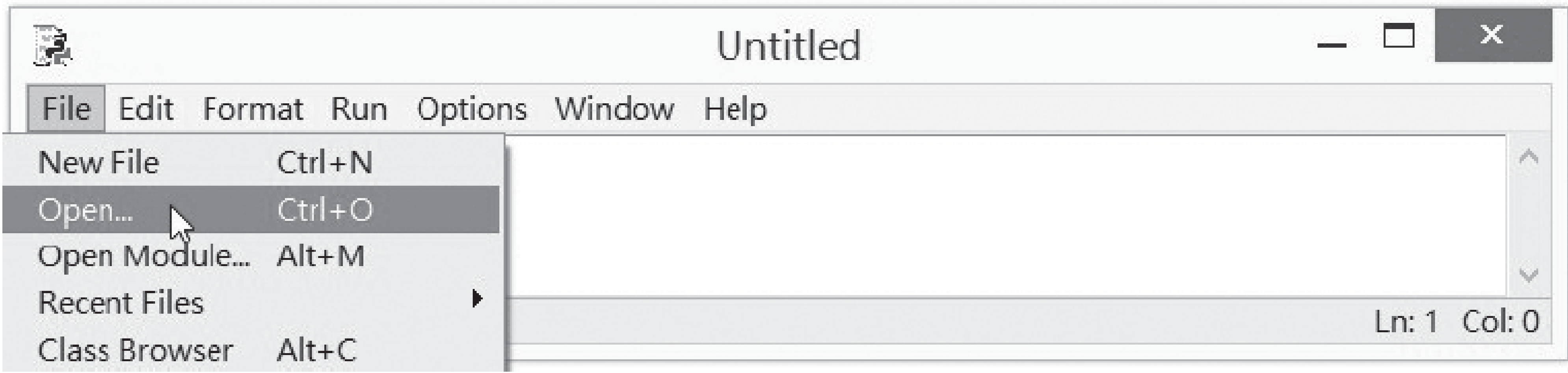
执行后，在原先的 Python Shell 窗口中可以看到执行结果。



学习到此，恭喜你已经成功地建立了一个 Python 文件，同时执行成功了。

1-9-4　打开文件

假设已经离开 ch1_1.py 文件，未来想要打开这个程序文件，可以执行 File → Open 命令。



然后会出现“打开文件”对话框，选择要打开的文件即可。

1-10　程序注释

程序注释的主要功能是让程序可读性更高，更容易了解。在企业工作中，一个实用的程序可以很轻易超过几千或上万行，此时可能需设计好几个月，给程序加上程序注释，可方便自己或他人了解程序内容。

1-10-1　注释符号

不论是使用 Python Shell 直译器或是 Python 程序文件中，“#”符号右边的文字，都称为程序注释，Python 语言的直译器会忽略此符号右边的文字。可参考下列实例。

实例 1：在 Python Shell 窗口注释的应用 1，注释可以放在程序语句的右边。

```
>>> print("Python数据科学零基础一本通")   # 打印本书名称
Python数据科学零基础一本通
>>> |
```

实例 2：在 Python Shell 窗口注释的应用 2，注释可以放在程序语句的最左边。

```
>>> # 打印本书名称
>>> print("Python数据科学零基础一本通")
Python数据科学零基础一本通
>>> |
```

程序实例 ch1_2.py：重新设计 ch1_1.py，为程序增加注释。

```
1   # ch1_2.py
2   print("Hello! Python")     # 打印字符串
```

注：Python 程序左边是没有行号的，上述行号是笔者为了读者阅读方便加上去的。

1-10-2　三个单引号或双引号

如果要进行大段落的注释，可以用三个单引号或双引号将注释文字包起来。

程序实例 ch1_3.py：以三个单引号当作注释。

```
1   '''
2   程序实例ch1_3.py
3   作者:洪锦魁
4   使用三个单引号当作注释
5   '''
6   print("Hello! Python")     # 打印字符串
```

上述前 5 行是程序注释。

程序实例 ch1_4.py：以三个双引号当作注释。

```
1   """
2   程序实例ch1_4.py
3   作者:洪锦魁
4   使用三个双引号当作注释
5   """
6   print("Hello! Python")     # 打印字符串
```

上述前 5 行是程序注释。

1-11　Python 彩蛋

Python 核心程序开发人员在软件内部设计了两个彩蛋，一个是搞笑网站，一个是经典名句又称 Python 之禅。这是在其他软件中没有见过的，非常有趣。

1. Python 之禅

在 Python Shell 环境下输入“import this”即可看到经典名句，其实这些经典名句也代表着研读 Python 的意境。

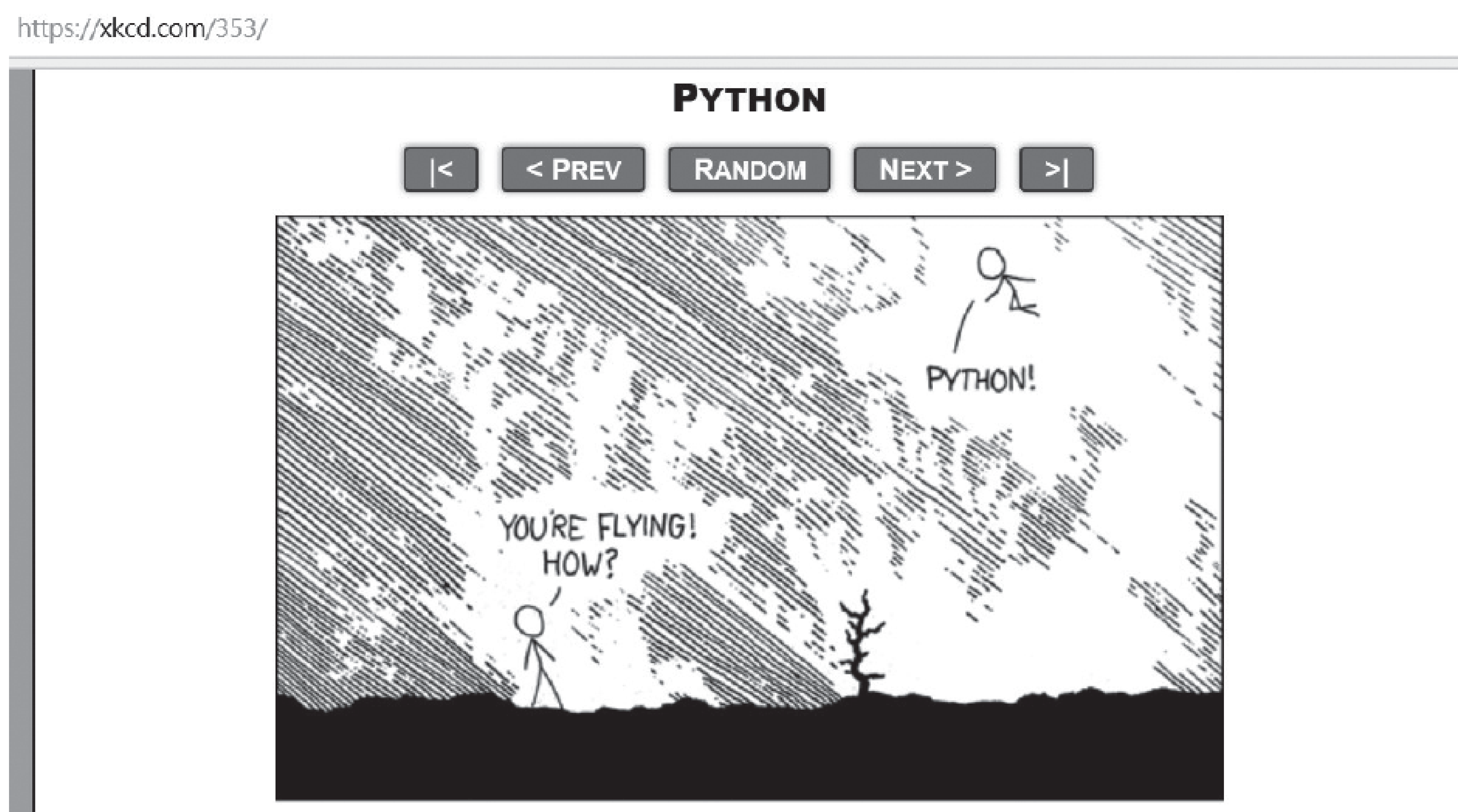
```
>>> import this
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

2. Python 搞笑网站

可以在 Python Shell 环境下输入“import antigravity”即可连接下列网址，读者可以欣赏有关 Python 的趣味内容。

https://xkcd.com/353/



习题

设计程序可以输出下列 3 行数据。

就读学校

年级

姓名

```
======================= RESTART: D:/Python/ex/ex1_1.py =======================
明志科技大学
一年级
洪锦魁
>>>
```

02

第 2 章

认识变量与基本数学运算

本章摘要

本章将从基本数学运算开始，一步一步讲解变量的使用与命名，接着介绍 Python 的算术运算。

2-1 用 Python 做计算

假设读者到麦当劳打工，一小时可以获得 120 元，如果想计算一天工作 8 小时，可以获得多少工资，可以用计算器执行 120×8，然后得到执行结果。在 Python Shell 中，可以使用下列方式计算。

```
>>> 120 * 8
960
>>>
```

如果一年实际工作天数是 300 天，可以用下列方式计算一年所得。

```
>>> 120 * 8 * 300
288000
>>>
>>> |
```

如果读者一个月的花费是 9000 元，可以用下列方式计算一年可以存储多少钱。

```
>>> 9000 * 12
108000
>>> 288000 - 108000
180000
>>>
```

上述先计算一年的花费，再用一年的收入减去一年的花费，可以得到所存储的金额。本章将一步一步推导应如何以程序思想，处理一般的运算问题。

2-2 认识变量

2-2-1 基本概念

变量是一个暂时存储数据的地方，对于 2-1 节的内容而言，如果时薪从 120 元调整到 125 元，想要重新计算一年可以存储多少钱，将发现所有的计算需要重新开始。为了解决这个问题，可以考虑将时薪设为一个变量，未来如果调整薪资，直接更改变量内容即可。

在 Python 中可以用“=”设置变量的内容，在这个实例中，建立了一个变量 x，然后用下列方式设置时薪。

```
>>> x = 120
>>>
```

如果想要用 Python 列出时薪，可以使用 print( ) 函数。

```
>>> print(x)
120
>>>
```

如果时薪从 120 元调整到 125 元，那么可以用下列方式表达。

```
>>> x = 125
>>> print(x)
125
>>>
```

注　在 Python Shell 环境，也可以直接输入变量名称，即可获得执行结果。

```
>>> x = 125
>>> x
125
>>>
```

一个程序中是可以使用多个变量的，如果想计算一天工作 8 小时，一年工作 300 天，可以赚多少钱，假设用变量 y 表示一年工作所赚的钱，可以用下列方式计算。

```
>>> x = 125
>>> y = x * 8 * 300
>>> print(y)
300000
>>>
```

如果每个月花费是 9000 元，使用变量 z 表示每个月的花费，可以用下列方式计算每年的花费，使用 a 表示每年的花费。

```
>>> z = 9000
>>> a = z * 12
>>> print(a)
108000
>>>
```

如果想计算每年可以存储多少钱，使用 b 表示每年所存储的钱，可以使用下列方式计算。

```
>>> x = 125
>>> y = x * 8 * 300
>>> z = 9000
>>> a = z * 12
>>> b = y - a
>>> print(b)
192000
>>>
```

上述语句顺利地使用 Python Shell 计算了每年可以存储多少钱，可是上述使用 Python Shell 做运算潜藏的最大问题是，只要过了一段时间，我们可能忘记当初所有设置的变量是代表什么意义。因此在设计程序时，如果可以为变量取个有意义的名称，未来看到程序时，可以比较容易记得。下列是笔者重新设计的变量名称。

时薪：hourly_salary，用此变量代替 x，即每小时的薪资。

年薪：annual_salary，用此变量代替 y，即一年工作所赚的钱。

月支出：monthly_fee，用此变量代替 z，即每个月的花费。

年支出：annual_fee，用此变量代替 a，即每年的花费。

年存储：annual_savings，用此变量代替 b，即每年所存储的钱。

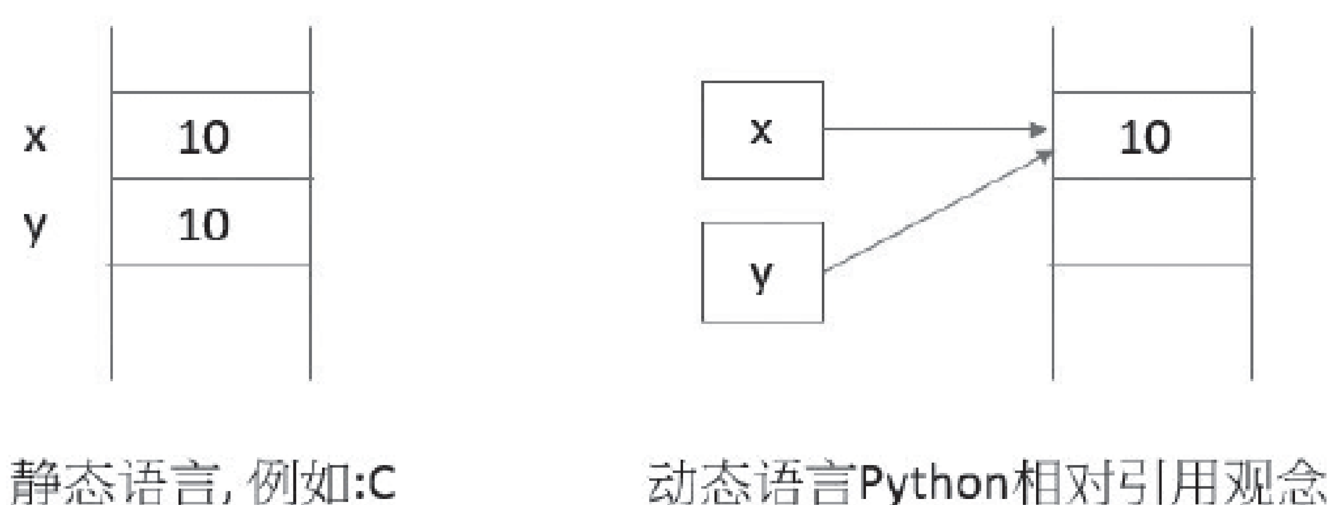
如果现在使用上述变量重新设计程序，可以得到下列结果。

```
>>> hourly_salary = 125
>>> annual_salary = hourly_salary * 8 * 300
>>> monthly_fee = 9000
>>> annual_fee = monthly_fee * 12
>>> annual_savings = annual_salary - annual_fee
>>> print(annual_savings)
192000
>>>
```

相信经过上述说明，读者应该了解变量的基本意义了。

2-2-2　认识变量的地址

Python 是一种动态语言，它处理变量的过程与一般静态语言不同。对于静态语言而言，例如 C、C++，当声明变量时内存就会预留空间存储此变量的内容，例如，若声明与定义 x=10, y=10 时，内存内容如下方左图所示。



静态语言，例如:C　　动态语言Python相对引用观念

对于 Python 而言，变量所使用的是参照（reference）地址的观念，设置一个变量 x 等于 10 时，Python 会在内存某个地址存储 10，此时我们建立的变量 x 好像是一个标志（tags），标志内容是存储 10 的内存地址。如果有另一个变量 y 也是 10，则变量 y 的标志内容也是存储 10 的内存地址，如上方右图所示。

使用 Python 可以使用 id( ) 函数获得变量的地址，可参考下列语法。

实例：列出变量的地址，相同内容的变量会有相同的地址。

```
>>> x = 10
>>> y = 10
>>> z = 20
>>> id(x)
1614727440
>>> id(y)
1614727440
>>> id(z)
1614727600
```

2-3 认识程序的意义

延续上一节的实例，如果时薪改变、工作天数改变或每个月的花费改变，所有输入与运算都要重新开始，而且每次都要重新输入程序代码，这是一件很费劲的事，同时很可能会输入错误，为了解决这个问题，可以使用 Python Shell 打开一个文件，将上述运算存储在文件内，这个文件就是所谓的程序。未来有需要时，再打开重新运算即可。

程序实例 ch2_1.py：使用程序计算每年可以存储多少钱，下面是整个程序设计。

```
1  # ch2_1.py
2  hourly_salary = 125
3  annual_salary = hourly_salary * 8 * 300
4  monthly_fee = 9000
5  annual_fee = monthly_fee * 12
6  annual_savings = annual_salary - annual_fee
7  print(annual_savings)
```

执行结果

```
====================== RESTART: D:\Python\ch2\ch2_1.py ======================
192000
```

未来时薪改变、工作天数改变或每个月的花费改变时，只要适度修改变量内容，就可以获得正确的执行结果。

2-4 认识注释的意义

程序 ch2_1.py 中尽管已经为变量设置了有意义的名称，但时间一久，常常还是会忘记各个指令的内涵。所以笔者建议，设计程序时，应适度地为程序代码加上注释。在 1-10 节已经讲解了注释的方法，下面将直接以实例说明。

程序实例 ch2_2.py：重新设计程序 ch2_1.py，为程序代码加上注释。

```
1  # ch2_2.py
2  hourly_salary = 125                           # 设置时薪
3  annual_salary = hourly_salary * 8 * 300       # 计算年薪
4  monthly_fee = 9000                            # 设置每月花费
5  annual_fee = monthly_fee * 12                 # 计算每年花费
6  annual_savings = annual_salary - annual_fee   # 计算每年存储金额
7  print(annual_savings)                         # 列出每年存储金额
```

执行结果 与 ch2_1.py 相同。

相信经过上述注释后，即使再过 10 年，只要一看到程序也可轻松了解整个程序的意义。

2-5 Python 变量与其他程序语言的差异

许多程序语言变量在使用前需要先声明，Python 对于变量的使用则是可以在需要时，再直接设置使用。有些程序语言在声明变量时，需要设置变量的数据类型，Python 则不需要设置，它会针对变量值的内容自行设置数据类型。

2-6 变量的命名原则

Python 对于变量的命名，在使用时有一些规则要遵守，否则会造成程序错误。

（1）必须由英文字母、_（下画线）或中文开头，建议使用英文字母。

（2）变量名称只能由英文字母、数字、_（下画线）或中文组成。

（3）英文字母大小写是敏感的，例如，Name 与 name 被视为不同变量名称。

（4）Python **系统保留字**（或称**关键词**）不可当作变量名称，会让程序产生错误，Python **内建函数名称**不建议当作变量名称。

注：虽然变量名称可以用中文，不过笔者不建议使用中文，将来可能会有兼容性的问题。

下列是不可当作变量名称的 Python **系统保留字**。

and	as	assert	break	class	continue
def	del	elif	else	except	False
finally	for	from	global	if	import
in	is	lambda	none	nonlocal	not
or	pass	raise	return	True	try
while	with	yield			

下列是不建议当作变量名称的 Python 系统**内建函数**，若是不小心将系统内建函数名称当作变量，程序本身不会错误，但是原函数功能会丧失。

abs( )	all( )	any( )	apply( )	basestring( )
bin( )	bool( )	buffer( )	bytearray( )	callable( )
chr( )	classmethod( )	cmp( )	coerce( )	compile( )
complex( )	delattr( )	dict( )	dir( )	divmod( )
enumerate( )	eval( )	execfile( )	file( )	filter( )
float( )	format( )	frozenset( )	getattr( )	globals( )
hasattr( )	hash( )	help( )	hex( )	id( )
input( )	int( )	intern( )	isinstance( )	issubclass( )
iter( )	len( )	list( )	locals( )	long( )
map( )	max( )	memoryview( )	min( )	next( )
object( )	oct( )	open( )	ord( )	pow( )
print( )	property( )	range( )	raw_input( )	reduce( )
reload( )	repr( )	reversed( )	round( )	set( )
setattr( )	slice( )	sorted( )	staticmethod( )	str( )
sum( )	super( )	tuple( )	type( )	unichr( )
unicode( )	vars( )	xrange( )	zip( )	_import( )

实例 1：下列是一些不合法的变量名称。

```
sum,1          # 变量不可有 ","
3y             # 变量不可由阿拉伯数字开头
x$2            # 变量不可有 "$" 符号
and            # 这是系统保留字不可当作变量名称
```

实例 2：下列是一些合法的变量名称。

SUM

_fg

x5

总和

实例 3：下列 3 个代表不同的变量。

SUM

Sum

sum

2-7 基本数学运算

2-7-1 四则运算

Python 的四则运算是指加（+）、减（-）、乘（*）和除（/）。

实例 1：下列是加法与减法运算实例。

```
>>> x = 5 + 6            # 将5加6设置给变量x
>>> print(x)
11
>>> y = x - 10           # 将x减10设置给变量y
>>> print(y)
1
>>>
```

实例 2：乘法与除法运算实例。

```
>>> x = 5 * 9            # 将5乘9设置给变量x
>>> print(x)
45
>>> y = 9 / 5            # 将9除以5设置给变量y
>>> print(y)
1.8
>>>
```

2-7-2 余数和整除

余数（mod）所使用的符号是“%”，可计算出除法运算中的余数。整除所使用的符号是“//”，是指除法运算中只保留整数部分。

实例：余数和整除运算实例。

```
>>> x = 9 % 5            # 将9除以5的余数设置给变量x
>>> print(x)
4
>>> y = 9 // 2           # 将9除以2的整数结果设置给变量y
>>> print(y)
4
>>>
```

其实在程序设计中求余数是非常有用的，例如，如果要判断数字是奇数或偶数可以用 %，例如

"num % 2"，如果 num 是奇数，所得结果是 1；如果 num 是偶数，所得结果是 0。当读者学会更多指令后，笔者会做更多的应用说明。

2-7-3　次方

次方的符号是“**”。

实例：平方、次方的运算实例。

```
>>> x = 3 ** 2          # 将3的平方设置给变量x
>>> print(x)
9
>>> y = 3 ** 3          # 将3的3次方设置给变量y
>>> print(y)
27
>>>
```

2-7-4　Python 语言控制运算的优先级

Python 语言碰上计算式同时出现在一个指令内时，除了括号“( )”内部运算最优先外，其余计算优先次序如下。

（1）次方；

（2）乘法、除法、求余数（%）、求整数（//），彼此依照出现顺序运算；

（3）加法、减法，彼此依照出现顺序运算。

实例：Python 语言控制运算的优先级的应用。

```
>>> x = ( 5 + 6 ) * 8 - 2
>>> print(x)
86
>>> y = 5 + 6 * 8 - 2
>>> print(y)
51
>>> z = 2 * 3**3 * 2
>>> print(z)
108
```

2-8　指派运算符

常见的指派运算符如下。

运算符	实例	说明
+=	a += b	a = a + b
-=	a -= b	a = a - b
*=	a *= b	a = a * b
/=	a /= b	a = a / b
%=	a %= b	a = a % b
//=	a //= b	a = a // b
**=	a **= b	a = a ** b

实例：指派运算符的实例说明。

```
>>> x = 10
>>> x += 5
>>> print(x)
15
>>> x = 10
>>> x -= 5
>>> print(x)
5
>>> x = 10
>>> x *= 5
>>> print(x)
50
>>> x = 10
>>> x /= 5
>>> print(x)
2.0
>>> x = 10
>>> x %= 5
>>> print(x)
0
>>> x = 10
>>> x //= 5
>>> print(x)
2
>>> x = 10
>>> x **= 5
>>> print(x)
100000
>>>
```

2-9 Python 等号的多重指定使用

使用 Python 时，可以一次设置多个变量等于某一数值。

实例 1：设置多个变量等于某一数值的应用。

```
>>> x = y = z = 10
>>> print(x)
10
>>> print(y)
10
>>> print(z)
10
>>>
```

Python 也允许多个变量同时指定不同的数值。

实例 2：设置多个变量，每个变量有不同值。

```
>>> x, y, z = 10, 20, 30
>>> print(x, y, z)
10 20 30
>>>
```

当执行上述多重设置变量值后，甚至可以执行更改变量内容。

实例 3：将两个变量内容交换。

```
>>> x, y = 10, 20
>>> print(x, y)
10 20
>>> x, y = y, x
>>> print(x, y)
20 10
>>>
```

上述原先 x, y 分别设为 10, 20，但是经过多重设置后变为 20, 10。其实可以使用多重指定更灵活地应用 Python，在 2-7-2 节有求商和余数的实例，可以使用 divmod( ) 函数一次获得商和余数，可参考下列实例。

```
>>> x = 9 // 5           # 将9除以5的整数给变量x
>>> print(x)
1
>>> y = 9 % 5            # 将9除以5的余数给变量y
>>> print(y)
4
>>> z = divmod(9,5)      # 一次获得商与余数
>>> print(z)
(1, 4)
>>> x,y = z
>>> print(x)
1
>>> print(y)
4
>>>
```

上述使用了 divmod（9,5）方法一次获得了元组值（1,4），第 8 章会介绍元组，然后使用多重指定将此元组（1,4）分别设置给 x 和 y 变量。

2-10 删除变量

程序设计时，如果某个变量不再需要，可以使用 del 指令将此变量删除，相当于可以收回原变量所占的内存空间，以节省内存空间。删除变量的格式如下：

del 变量名称

实例：验证变量名称回收后，将无法再使用。此例中尝试输出已删除的变量，然后程序出现错误消息。

```
>>> x = 10
>>> print(x)
10
>>> del x
>>> print(x)
Traceback (most recent call last):
  File "<pyshell#157>", line 1, in <module>
    print(x)
NameError: name 'x' is not defined
>>>
```


2-11 Python 的断行

2-11-1 一行有多个语句

在 Python 中允许一行有多个，彼此用“;”隔开即可，尽管 Python 有提供此功能，不过笔者不鼓励如此撰写程序代码。

程序实例 ch2_3.py：一行有多个语句的实例。

```
1  # ch2_3.py
2  x = 10
3  print(x)
4  y = 20;print(y)          # 一行有两个语句，不过不鼓励这种写法
```

执行结果

```
====================== RESTART: D:\Python\ch2\ch2_3.py ======================
10
20
```

2-11-2　将一个语句分成多行

在设计大型程序时，常会碰上一个语句很长，需要分成两行或更多行撰写，此时可以在语句后面加上“\”符号，Python 解释器会将下一行的语句视为这一行的语句。特别注意，在“\”符号右边不可以加上任何符号或文字，即使是注释符号也是不允许的。

另外，也可以在语句内使用小括号，如果使用小括号，就可以在语句右边加上注释符号。

程序实例 ch2_4.py：将一个语句分成多行的应用。

```
1  # ch2_4.py
2  a = b = c = 10
3  x = a + b + c + 12
4  print(x)
5  # 续行方法1
6  y = a +\
7      b +\
8      c +\
9      12
10 print(y)
11 # 续行方法2
12 z = ( a +      # 此处可以加上注释
13       b +
14       c +
15       12 )
16 print(z)
```

执行结果

```
====================== RESTART: D:\Python\ch2\ch2_4.py ======================
42
42
42
```

2-12　专题——复利计算 / 计算圆面积与圆周长

2-12-1　银行存款复利的计算

程序实例 ch2_5.py：银行存款复利的计算。假设目前银行年利率是 1.5%，复利公式如下：

本金和 ＝ 本金 ×（ 1 ＋ 年利率 ）^{n}　　　　　# n是年

现有一笔 5 万元存款，请计算 5 年后的本金和。

```
1  # ch2_5.py
2  money = 50000 * ( 1 + 0.015 ) ** 5
3  print("本金和是")
4  print(money)
```

执行结果

```
===================== RESTART: D:\Python\ch2\ch2_5.py =====================
本金和是
53864.20019421873
```

2-12-2 计算圆面积与周长

程序实例 ch2_6.py：假设圆半径是 5cm，圆面积与圆周长计算公式分别如下：

圆面积 ＝ PI×r×r　　　　　# PI ＝ 3.14159, r是半径

圆周长 ＝ 2×PI×r

```
1  # ch2_6.py
2  PI = 3.14159
3  r = 5
4  print("圆面积:单位是平方厘米")
5  area = PI * r * r
6  print(area)
7  circurference = 2 * PI * r
8  print("圆周长:单位是厘米")
9  print(circurference)
```

执行结果

```
===================== RESTART: D:\Python\ch2\ch2_6.py =====================
圆面积:单位是平方厘米
78.53975
圆周长:单位是厘米
31.4159
>>>
```

在程序语言的设计中，有一个概念是常量（named constant），这种常量是不可更改内容的。上述计算圆面积或圆周长所使用的 PI 是圆周率，这是一个固定的值，由于 Python 语言没有提供此常量（names constant）的语法，上述程序笔者用大写 PI 当作常量的变量，这是一种习惯，未来读者可以用这种方式处理固定内容的变量。

习题

1. 请重新设计 ch2_1.py，将打工时薪改为 150 元。(2-1 ~ 2-3 节)

```
===================== RESTART: D:/Python/ex/ex2_1.py =====================
每年存款金额
252000
```

2. 重新设计 ch2_5.py，假设是单利率，5 年期间可以领多少利息？（2-5 ～ 2-7 节）

```
======================= RESTART: D:/Python/ex/ex2_2.py =======================
利息总和
3750.0
```

3. 重新设计 ch2_5.py，假设期初本金是 100 000 元，年利率是 2%，这是复利计算，请问 10 年后本金总和是多少？（2-5 ～ 2-12 节）

```
======================= RESTART: D:\Python\ex\ex2_3.py =======================
10年后本金和
121899.44199947573
```

4. 一个幼儿园买了 100 个苹果给学生当营养午餐，学生人数是 23 人，每个人午餐可以吃一个，请问这些苹果可以吃几天？第几天苹果会不够供应？同时列出缺少了几个。（2-5 ～ 2-12 节）

```
======================= RESTART: D:/Python/ex/ex2_4.py =======================
苹果可以吃的天数
4
第几天产生苹果供应不足
5
不足数量
15
```

5. 地球和月球的距离是 384 400 千米，假设火箭飞行速度是每分钟 400 千米，请问从地球飞到月球需要多少分钟？（2-5 ～ 2-12 节）

```
======================= RESTART: D:/Python/ex/ex2_5.py =======================
地球到月球所需分钟总数
961.0
```

6. 假设圆柱半径是 20 厘米，高度是 30 厘米，请计算此圆柱的体积。圆柱体积计算公式是：圆面积 × 圆柱高度。（2-5 ～ 2-12 节）

```
======================= RESTART: D:/Python/ex/ex2_6.py =======================
圆柱体积:单位是立方厘米
37699.08
```

7. 圆周率 PI 是一个数学常数，常常使用希腊字母表示，它的物理意义是圆的周长和直径的比率。历史上第一个无穷级数公式称为**莱布尼茨**公式，它的计算公式如下：（2-5 ～ 2-12 节）

$$\mathrm{PI}=4\times\left(1-\frac{1}{3}+\frac{1}{5}-\frac{1}{7}+\frac{1}{9}-\frac{1}{11}+\cdots\right)$$

请分别计算下列级数的执行结果。

（1）$\mathrm{PI}=4\times\left(1-\frac{1}{3}+\frac{1}{5}-\frac{1}{7}+\frac{1}{9}\right)$

（2）$\mathrm{PI}=4\times\left(1-\frac{1}{3}+\frac{1}{5}-\frac{1}{7}+\frac{1}{9}-\frac{1}{11}\right)$

（3）$\mathrm{PI}=4\times\left(1-\frac{1}{3}+\frac{1}{5}-\frac{1}{7}+\frac{1}{9}-\frac{1}{11}+\frac{1}{13}\right)$

注　上述级数如果要收敛到我们熟知的 3.14159 需要相当长的级数计算。

```
======================= RESTART: D:\Python\ex\ex2_7.py =======================
PI的值4 * (1 - 1/3 + 1/5 - 1/7 + 1/9)
3.3396825396825403
PI的值4 * (1 - 1/3 + 1/5 - 1/7 + 1/9 - 1/11)
2.9760461760461765
PI的值4 * (1 - 1/3 + 1/5 - 1/7 + 1/9 - 1/11 + 1/13)
3.2837384837384844
>>>
```

莱布尼茨（Leibniz，1646—1716）是德国人，在世界数学舞台上占有一席之地，他本人另一个职业是律师，许多数学公式都是他在各大城市通勤期间完成的。数学历史上有一个两派说法的无解公案，有人认为他是微积分的发明人，也有人认为发明人是牛顿（Newton）。

8. 尼拉卡莎级数也是应用于计算圆周率 PI 的级数，此级数收敛的速度比莱布尼茨级数更好，更适合于用来计算 PI，它的计算公式如下：

$$PI=3+\frac{4}{2\times3\times4}-\frac{4}{4\times5\times6}+\frac{4}{6\times7\times8}-\cdots$$

请分别计算下列级数的执行结果。

（a） $PI=3+\frac{4}{2\times3\times4}-\frac{4}{4\times5\times6}+\frac{4}{6\times7\times8}-\cdots$

（b） $PI=3+\frac{4}{2\times3\times4}-\frac{4}{4\times5\times6}+\frac{4}{6\times7\times8}-\frac{4}{8\times9\times10}\cdots$

```
======================= RESTART: D:/Python/ex/ex2_8.py =======================
PI的值3 + 4/(2*3*4) - 4/(4*5*6) + 4/(6*7*8)
3.145238095238095
PI的值3 + 4/(2*3*4) - 4/(4*5*6) + 4/(6*7*8) - 4/(8*9*10)
3.1396825396825396
```

03

第 3 章

Python 的基本数据类型

本章摘要

3-1 type( ) 函数
3-2 数值数据类型
3-3 布尔值数据类型
3-4 字符串数据类型
3-5 字符串与字符
3-6 bytes 数据
3-7 专题——地球到月球时间计算 / 计算坐标轴两点之间的距离

Python 的基本数据类型有下列几种。

（1）**数值数据类型**（numeric type）：常见的数值数据又可分成**整数**（int）、**浮点数**（float）、**复数**（complex number）（**不常用**所以不在本书讨论范围）。

（2）**布尔值**（Boolean）数据类型：也可归为数值数据类型。

（3）**文字序列**类型（text sequence type）：也就是字符串（string）数据类型。

（4）**字符组**（bytes，有的书称**字节**）数据类型：这是二进制的数据类型，长度是 8 位。

（5）**序列**类型（sequence type）：list（第 6 章说明）、tuple（第 8 章说明）。

（6）**对映**类型（mapping type）：dict（第 9 章说明）。

（7）**集合**类型（set type）：集合 set（第 10 章说明）、冻结集合 frozenset。

3-1 type( ) 函数

在正式介绍 Python 的数据类型前，笔者想介绍一下 type( ) 函数，这个函数可以列出变量的数据类型类别。这个函数在读者未来进入 Python 实战时非常重要，因为变量在使用前不需要声明，同时在程序设计过程中变量的数据类型会改变，我们常常需要使用此函数判断目前的变量数据类型。或是在进阶 Python 应用中，会调用一些方法（method），这些方法会返回一些数据，可以使用 type( ) 获得所返回的数据类型。

程序实例 ch3_1.py：列出数值变量的数据类型。

```
1   # ch3_1.py
2   x = 10
3   y = x / 3
4   print(x)
5   print(type(x))
6   print(y)
7   print(type(y))
```

执行结果

```
===================== RESTART: D:/Python/ch3/ch3_1.py =====================
10
<class 'int'>
3.3333333333333335
<class 'float'>
```

从上述执行结果可以看到，变量 x 的内容是 10，数据类型是整数（int）。变量 y 的内容是 3.33…3，数据类型是浮点数（float）。下一节会说明为何是这样。

3-2 数值数据类型

3-2-1 整数 int

整数的英文是 integer，在计算机程序语言中一般用 int 表示。如果读者学过其他计算机语言，

在介绍整数时一定会告诉你，该计算机语言使用了多少空间存储整数，所以设计程序时整数的大小必须是在某一区间，否则会有**溢位**（overflow）造成数据不正确。例如，如果存储整数的空间是 32 位，则整数大小为 -2 147 483 648 ～ 2 147 483 647。在 Python 2.x 版时代，整数被限制在 32 位，另外还有长整数 long，空间大小是 64 位，所以可以存储的数值更大，达到 -9 223 372 036 854 775 808 ~ 9 223 372 036 854 775 807。在 Python 3 中已经将整数可以存储空间大小的限制取消了，所以没有 long 了，也就是说 int 可以是任意大小的数值。

英文 googol 是指自然数 10^{100}，计算机是用 $1e^{100}$ 显示，这是 1938 年美国数学家**爱德华・卡斯纳**（Edward Kasner）9 岁的侄子**米尔顿・西罗蒂**（Milton Sirotta）所创造的。下列是笔者尝试使用整数 int 显示此 googol 值。

```
>>> googol = 10 ** 100
>>> googol
10000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000
```

3-2-2　浮点数

浮点数的英文是 float，既然整数大小没有限制，浮点数大小当然也没有限制。在 Python 语言中，带有小数点的数字称为**浮点数**。例如：

```
x = 10.3
```

表示 x 是浮点数。

3-2-3　基本数值数据的使用

Python 在声明变量时可以不用设置这个变量的数据类型，未来如果这个变量内容是放整数，这个变量就是整数（int）数据类型，如果这个变量内容是放浮点数，这个变量就是浮点数数据类型。整数与浮点数最大的区别是，整数不含小数点，浮点数含小数点。

程序实例 ch3_2.py：测试浮点数。

```
1  # ch3_2.py
2  x = 10.0
3  print(x)
4  print(type(x))
```

执行结果

```
===================== RESTART: D:/Python/ch3/ch3_2.py =====================
10.0
<class 'float'>
```

在程序实例 ch3_1.py 中，x 变量的值是“10”，列出 x 变量是**整数变量**，在这个实例中，x 变量的值是“10.0”，列出 x 变量是**浮点数变量**。

3-2-4　整数与浮点数的运算

Python 程序设计时不相同的数据类型也可以执行运算，程序设计时常常会发生整数与浮点数之

间的数据运算，Python 具有简单的自动转换能力，在计算时会将整数转换为浮点数再执行运算。

程序实例 ch3_3.py：不同数据类型的运算。

```
1  # ch3_3.py
2  x = 10
3  y = x + 5.5
4  print(x)
5  print(type(x))
6  print(y)
7  print(type(y))
```

执行结果

```
===================== RESTART: D:/Python/ch3/ch3_3.py =====================
10
<class 'int'>
15.5
<class 'float'>
```

上述变量 y，由于是整数与浮点数的加法，所以结果是浮点数。此外，某一个变量如果是整数，但是如果最后所存储的值是浮点数，Python 也会将此变量转成浮点数。

程序实例 ch3_4.py：整数转换成浮点数的应用。

```
1  # ch3_4.py
2  x = 10
3  print(x)
4  print(type(x))        # 加法前列出x数据类型
5  x = x + 5.5
6  print(x)
7  print(type(x))        # 加法后列出x数据类型
```

执行结果

```
===================== RESTART: D:/Python/ch3/ch3_4.py =====================
10
<class 'int'>
15.5
<class 'float'>
```

原先变量 x 所存储的值是整数，所以列出的是整数。后来存储了浮点数，所以列出的是浮点数。

3-2-5 二进制整数与函数 bin( )

可以用二进制方式代表整数，Python 中定义凡是以 0b 开头的数字，代表这是二进制的整数。

bin( ) 函数可以将一般整数数字转换为二进制。

程序实例 ch3_5.py：将十进制数值与二进制数值互转的应用。

```
1  # ch3_5.py
2  x = 0b1101            # 这是二进制整数
3  print(x)              # 列出十进制的结果
4  y = 13                # 这是十进制整数
5  print(bin(y))         # 列出转换成二进制的结果
```

执行结果

```
====================== RESTART: D:/Python/ch3/ch3_5.py ======================
13
0b1101
```

3-2-6　八进制整数与函数 oct( )

可以用八进制方式代表整数，Python 中定义凡是以 0o 开头的数字，代表这是八进制的整数。

oct( ) 函数可以将一般数字转换为八进制。

程序实例 ch3_6.py：将十进制数值与八进制数值互转的应用。

```
1  # ch3_6.py
2  x = 0o57                 # 这是八进制整数
3  print(x)                 # 列出十进制的结果
4  y = 47                   # 这是十进制整数
5  print(oct(y))            # 列出转换成八进制的结果
```

执行结果

```
====================== RESTART: D:/Python/ch3/ch3_6.py ======================
47
0o57
```

3-2-7　十六进制整数与函数 hex( )

可以用十六进制方式代表整数，Python 中定义凡是以 0x 开头的数字，代表这是十六进制的整数。

hex( ) 函数可以将一般数字转换为十六进制。

程序实例 ch3_7.py：将十进制数值与八进制数值互转的应用。

```
1  # ch3_7.py
2  x = 0x5D                 # 这是十六进制整数
3  print(x)                 # 列出十进制的结果
4  y = 93                   # 这是十进制整数
5  print(hex(y))            # 列出转换成十六进制的结果
```

执行结果

```
====================== RESTART: D:/Python/ch3/ch3_7.py ======================
93
0x5d
```

3-2-8　强制数据类型的转换

有时候设计程序时，可以自行强制使用下列函数，转换变量的数据类型。

int( )：将数据类型强制转换为整数。

float( )：将数据类型强制转换为浮点数。

程序实例 ch3_8.py：将浮点数强制转换为整数的运算。

```
1  # ch3_8.py
2  x = 10.5
3  print(x)
4  print(type(x))        # 加法前列出x数据类型
5  y = int(x) + 5
6  print(y)
7  print(type(y))        # 加法后列出y数据类型
```

执行结果

```
===================== RESTART: D:/Python/ch3/ch3_8.py =====================
10.5
<class 'float'>
15
<class 'int'>
```

程序实例 ch3_9.py：将整数强制转换为浮点数的运算。

```
1  # ch3_9.py
2  x = 10
3  print(x)
4  print(type(x))        # 加法前列出x数据类型
5  y = float(x) + 10
6  print(y)
7  print(type(y))        # 加法后列出y数据类型
```

执行结果

```
===================== RESTART: D:/Python/ch3/ch3_9.py =====================
10
<class 'int'>
20.0
<class 'float'>
```

3-2-9　数值运算常用的函数

下列是数值运算时常用的函数。

abs( )：计算绝对值。

pow (x,y)：返回 x 的 y 次方。

round( )：这是采用运算法则的 Bankers Rounding 概念，如果处理位数左边是**奇数则使用四舍五入**，如果处理位数左边是**偶数则使用五舍六入**，例如，round(1.5)=2，round(2.5)=2。

处理小数时，第 2 个参数代表取到小数第几位，1 代表取到小数第 1 位。根据保留小数位的后两位，采用 "50" 舍去，"51" 进位，例如，round(2.15,1)=2.1，round(2.25,1)=2.2，round(2.151,1)=2.2，round(2.251,1)=2.3。

程序实例 ch3_10.py：abs( )、pow( )、round( )、round(x,n）函数的应用。

```
1  # ch3_10.py
2  x = -10
3  print("以下输出abs()函数的应用")
4  print(x)              # 输出x变量
5  print(abs(x))         # 输出abs(x)
6  x = 5
7  y = 3
8  print("以下输出pow()函数的应用")
9  print(pow(x, y))      # 输出pow(x,y)
10 x = 47.5
11 print("以下输出round(x)函数的应用")
12 print(x)              # 输出x变量
13 print(round(x))       # 输出round(x)
14 x = 48.5
15 print(x)              # 输出x变量
16 print(round(x))       # 输出round(x)
17 x = 49.5
18 print(x)              # 输出x变量
19 print(round(x))       # 输出round(x)
20 print("以下输出round(x,n)函数的应用")
21 x = 2.15
22 print(x)              # 输出x变量
23 print(round(x,1))     # 输出round(x,1)
24 x = 2.25
25 print(x)              # 输出x变量
26 print(round(x,1))     # 输出round(x,1)
27 x = 2.151
28 print(x)              # 输出x变量
29 print(round(x,1))     # 输出round(x,1)
30 x = 2.251
31 print(x)              # 输出x变量
32 print(round(x,1))     # 输出round(x,1)
```

执行结果

```
===================== RESTART: D:\Python\ch3\ch3_10.py =====================
以下输出abs()函数的应用
-10
10
以下输出pow()函数的应用
125
以下输出round(x)函数的应用
47.5
48
48.5
48
49.5
50
以下输出round(x,n)函数的应用
2.15
2.1
2.25
2.2
2.151
2.2
2.251
2.3
```

需留意的是，使用上述 abs( )、pow( ) 或 round( ) 函数，尽管可以得到运算结果，但是原先变量的值是没有改变的。

3-2-10　科学记数法

科学记数的概念如下，将一个数字转换成下列数学式：

$a \times 10^n$

a 是浮点数，例如，123456 可以表示为 1.23456 $\times 10^5$，以 10 为基底数我们用 E 或 e 表示，指数部分则转为一般数字，然后省略“×”符号，最后表达式如下：

1.23456E+5

或

1.23456e+5

如果是碰上小于 1 的数值，则 E 或 e 右边是负值“-”。例如，0.000123 转成科学记数法，最后表达式如下：

1.23E-4

或

1.23e-4

下列是示范输出。

```
>>> x = 1.23456E+5
>>> x
123456.0
>>> y = 1.23e-4
>>> y
0.000123
```

4-2-2 节和 4-2-3 节会介绍将一般数值转成科学记数法输出的方式，以及格式化输出方式。

3-3 布尔值数据类型

Python 的布尔值（Boolean）数据类型的值有两种，True（真）或 False（伪）。它的数据类型代号是 bool。布尔值一般应用在程序流程的控制中，特别是在条件表达式中，程序可以根据这个布尔值判断应该如何执行下一步工作。

程序实例 ch3_11.py：列出布尔值 True 与布尔值 False 的数据类型。

```
1  # ch3_11.py
2  x = True
3  print(x)
4  print(type(x))        # 列出x数据类型
5  y = False
6  print(y)
7  print(type(y))        # 列出y数据类型
```

执行结果

```
====================== RESTART: D:/Python/ch3/ch3_11.py ======================
True
<class 'bool'>
False
<class 'bool'>
```

如果将布尔值数据类型强制转换成整数，如果原值是 True，将得到 1；如果原值是 False，将得到 0。

程序实例 ch3_12.py：将布尔值强制转换为整数，同时列出转换的结果。

```
1  # ch3_12.py
2  x = True
3  print(int(x))
4  print(type(x))        # 列出x数据类型
5  y = False
6  print(int(y))
7  print(type(y))        # 列出y数据类型
```

执行结果

```
====================== RESTART: D:/Python/ch3/ch3_12.py ======================
1
<class 'bool'>
0
<class 'bool'>
```

在本章一开始说过，有时候也可以将布尔值当作数值数据，因为 True 会被视为 1，False 会被视为 0，可以参考下列实例。

程序实例 ch3_13.py：将布尔值与整数值相加，并观察最后变量数据类型，可以发现，最后的变量数据类型是整数值。

```
1  # ch3_13.py
2  xt = True
3  x = 1 + xt
4  print(x)
5  print(type(x))        # 列出x数据类型
6
7  yt = False
8  y = 1 + yt
9  print(y)
10 print(type(y))        # 列出y数据类型
```

执行结果

```
====================== RESTART: D:/Python/ch3/ch3_13.py ======================
2
<class 'int'>
1
<class 'int'>
```

此外，在程序设计中 False 值不一定是要经过条件判断是 False，才可以得到 False，下列情况也会被视为 False。

布尔值 False

整数 0

浮点数 0.0

空字符串 ''

空列表 [ ]

空元组 ( )

空字典 { }

空集合 set( )

None

至于其他的都会被视为 True。

3-4 字符串数据类型

字符串（string）数据是指两个单引号（'）之间或是两个双引号（"）之间任意个数字元符号的数据，它的数据类型代号是 str。在英文字符串的使用中常会发生某字中间有单引号的情况，其实这是文字的一部分，如下所示：

```
This is James's ball
```

如果用单引号去处理上述字符串将产生错误，如下所示：

```
>>> x = 'This is James's ball'
SyntaxError: invalid syntax
>>>
```

碰到这种情况，可以用双引号解决，如下所示：

```
>>> x = "This is James's ball"
>>> print(x)
This is James's ball
>>>
```

程序实例 ch3_14.py：使用单引号与双引号设置与输出字符串数据的应用。

```
1  # ch3_14.py
2  x = "DeepStone means Deep Learning"     # 双引号设置字符串
3  print(x)
4  print(type(x))                           # 列出x字符串数据类型
5  y = '深石数字 - 深度学习滴水穿石'        # 单引号设置字符串
6  print(y)
7  print(type(y))                           # 列出y字符串数据类型
```

执行结果

```
===================== RESTART: D:\Python\ch3\ch3_14.py =====================
DeepStone means Deep Learning
<class 'str'>
深石数字 - 深度学习滴水穿石
<class 'str'>
```

3-4-1 字符串的连接

数学的运算符“+”，可以进行两个字符串相加的操作，产生新的字符串。

程序实例 ch3_15.py：字符串连接的应用。

```
1  # ch3_15.py
2  num1 = 222
3  num2 = 333
4  num3 = num1 + num2
5  print("以下是数值相加")
6  print(num3)
7  numstr1 = "222"
8  numstr2 = "333"
9  numstr3 = numstr1 + numstr2
10 print("以下是由数值组成的字符串相加")
11 print(numstr3)
12 numstr4 = numstr1 + " " + numstr2
13 print("以下是由数值组成的字符串相加，同时中间加上一空格")
14 print(numstr4)
15 str1 = "DeepStone "
16 str2 = "Deep Learning"
17 str3 = str1 + str2
18 print("以下是一般字符串相加")
19 print(str3)
```

执行结果

```
====================== RESTART: D:\Python\ch3\ch3_15.py ======================
以下是数值相加
555
以下是由数值组成的字符串相加
222333
以下是由数值组成的字符串相加，同时中间加上一空格
222 333
以下是一般字符串相加
DeepStone Deep Learning
```

3-4-2　处理多于一行的字符串

程序设计时如果字符串长度多于一行，可以使用三个单引号（或是三个双引号）将字符串括起来即可。

程序实例 ch3_16.py：使用三个单引号处理多于一行的字符串。

```
1  # ch3_16.py
2  str1 = '''Silicon Stone Education is an unbiased organization
3  concentrated on bridging the gap ... '''
4  print(str1)
```

执行结果

```
====================== RESTART: D:/Python/ch3/ch3_16.py ======================
Silicon Stone Education is an unbiased organization
concentrated on bridging the gap ...
```

读者可以留意第 2 行 Silicon 左边的 3 个单引号和第 3 行末端的 3 个单引号，另外，上述第 2 行若是少了 "str1 = "，3 个单引号间的跨行字符串就变成了程序的注释。

3-4-3　转义字符

在字符串使用中，如果字符串内有一些特殊字符，例如单引号、双引号等，必须在此特殊字符前加上“\”（反斜杠），才可正常使用，这种含有“\”符号的字符称为**转义字符**（Escape Character）。

转义字符	Hex 值	意义	转义字符	Hex 值	意义
\'	27	单引号	\n	0A	换行
\"	22	双引号	\o		八进制表示
\\	5C	反斜杠	\r	0D	光标移至最左位置
\a	07	响铃	\x		十六进制表示
\b	08	Backspace 键	\t	09	Tab 键
\f	0C	换页	\v	0B	垂直定位

字符串使用中特别是碰到字符串含有单引号时，如果是使用单引号定义这个字符串，必须要使用此**转义字符**，才可以顺利显示，可参考 ch3_17.py 的第 3 行。如果是使用双引号定义字符串，则可以不必使用**转义字符**，可参考 ch3_17.py 的第 6 行。

程序实例 ch3_17.py：转义字符的应用，这个程序第 9 行增加了“\t”字符，所以“can't”跳到下一个 Tab 键位置输出。同时有“\n”字符，这是换行符号，所以“loving”跳到下一行输出。

```
1  # ch3_17.py
2  #以下输出使用单引号设置的字符串,需使用\'
3  str1 = 'I can\'t stop loving you.'
4  print(str1)
5  #以下输出使用双引号设置的字符串,不需使用\'
6  str2 = "I can't stop loving you."
7  print(str2)
8  #以下输出有\t和\n字符
9  str3 = "I \tcan't stop \nloving you."
10 print(str3)
```

执行结果

```
===================== RESTART: D:/Python/ch3/ch3_17.py =====================
I can't stop loving you.
I can't stop loving you.
I       can't stop
loving you.
```

3-4-4 str( ) 函数

str( ) 函数有如下几个用法。

（1）设置空字符串。

```
>>> x = str()                    # 设置空字符串
>>> x
''
>>> print(x)

>>>
```

（2）设置字符串。

```
>>> x = str('ABC')
>>> x
'ABC'
```

（3）强制将数值数据转换为字符串数据。

```
>>> x = 123
>>> y = str(x)
>>> y
'123'
```

程序实例 ch3_18.py：使用 str( ) 函数将数值数据强制转换为字符串的应用。

```
1  # ch3_18.py
2  num1 = 222
3  num2 = 333
4  num3 = num1 + num2
5  print("这是数值相加")
6  print(num3)
7  str1 = str(num1) + str(num2)
8  print("强制转换为字符串相加")
9  print(str1)
```

执行结果

```
===================== RESTART: D:\Python\ch3\ch3_18.py =====================
这是数值相加
555
强制转换为字符串相加
222333
```

上述字符串相加，读者可以想成是字符串连接，执行结果是一个字符串，所以上述执行结果 555 是数值数据，222333 则是一个字符串。

3-4-5　将字符串转换为整数

int( ) 函数可以将字符串转为整数，在未来的程序设计中也常会发生将字符串转换为整数数据，下面将直接以实例做说明。**注**：如果数字是非数字字符组成，会产生错误。

程序实例 ch3_19.py：将字符串数据转换为整数数据的应用。

```
1  # ch3_19.py
2  x1 = "22"
3  x2 = "33"
4  x3 = x1 + x2
5  print(x3)                  # 打印字符串相加
6  x4 = int(x1) + int(x2)
7  print(x4)                  # 打印整数相加
```

执行结果

```
===================== RESTART: D:/Python/ch3/ch3_19.py =====================
2233
55
```

上述执行结果 55 是数值数据，2233 则是一个字符串。

3-4-6　字符串与整数相乘产生字符串复制效果

在 Python 中允许将字符串与整数相乘，结果是字符串将重复该整数的次数。

程序实例 ch3_20.py：字符串与整数相乘的应用。

```
1  # ch3_20.py
2  x1 = "A"
3  x2 = x1 * 10
4  print(x2)                  # 打印字符串乘以整数
5  x3 = "ABC"
6  x4 = x3 * 5
7  print(x4)                  # 打印字符串乘以整数
```

执行结果

```
===================== RESTART: D:/Python/ch3/ch3_20.py =====================
AAAAAAAAAA
ABCABCABCABCABC
```

3-4-7 聪明地使用字符串加法和换行字符 \n

有时在设计程序时，想将字符串分行输出，可以使用字符串加法功能，在加法过程中加上换行字符“\n”即可产生字符串分行输出的结果。

程序实例 ch3_21.py：将数据分行输出的应用。

```
1  # ch3_21.py
2  str1 = "洪锦魁著作"
3  str2 = "HTML5+CSS3王者归来"
4  str3 = "Python数据科学零基础一本通"
5  str4 = str1 + "\n" + str2 + "\n" + str3
6  print(str4)
```

执行结果

```
====================== RESTART: D:\Python\ch3\ch3_21.py ======================
洪锦魁著作
HTML5+CSS3王者归来
Python数据科学零基础一本通
```

3-4-8 字符串前加 r

在使用 Python 时，如果在字符串前加上 r，可以防止转义字符被转义，可参考 3-4-3 节的转义字符表，相当于可以取消转义字符的功能。

程序实例 ch3_22.py：字符串前加上 r 的应用。

```
1  # ch3_22.py
2  str1 = "Hello!\nPython"
3  print("不含r字符的输出")
4  print(str1)
5  str2 = r"Hello!\nPython"
6  print("含r字符的输出")
7  print(str2)
```

执行结果

```
====================== RESTART: D:\Python\ch3\ch3_22.py ======================
不含r字符的输出
Hello!
Python
含r字符的输出
Hello!\nPython
```

3-5 字符串与字符

在 Python 中没有所谓的字符（character）数据，如果字符串含一个字符，我们称这是含一个字符的字符串。

3-5-1　ASCII 码

计算器内部最小的存储单位是位（bit），这个位只能存储 0 或 1。一个英文字符在计算器中是被存储成 8 个位的一连串 0 或 1 中，存储这个英文字符的编码称为 ASCII（American Standard Code for Information Interchange，美国信息交换标准程序代码），有关 ASCII 码的内容可以参考附录 E。

在这个 ASCII 表中由于是用 8 位定义一个字符，所以使用了 0 ～ 127 定义了 128 个字符，在这 128 个字符中有 33 个字符是无法显示的控制字符，其他则是可以显示的字符。不过有一些应用程序扩充了功能，让部分控制字符可以显示，例如，扑克牌花色、笑脸等。至于其他可显示字符有一些符号，例如 +、-、=、0 ～ 9、A ～ Z 或 a ～ z 等。这些符号每一个都有一个编码，我们称这个编码是 ASCII 码。

可以使用下列函数执行数据的转换。

chr (x)：返回函数 x 值的 ASCII 或 Unicode 字符。

例如，从 ASCII 表可知，字符 a 的 ASCII 码值是 97，可以使用下列方式打印出此字符。

```
>>> x = 97
>>> print(chr(x))
a
```

英文小写与英文大写的码值相差 32，可参考下列实例。

```
>>> x = 97
>>> x -= 32
>>> print(chr(x))
A
```

3-5-2　Unicode 码

计算机是美国发明的，因此 ASCII 码对于英语系国家的确很好用，但是地球是一个多种族的社会，存在几百种语言与文字，ASCII 所能容纳的字符是有限的，只要随便一个不同语系的外来词，例如 café，含重音字符就无法显示了，更何况有几万中文字或其他语系文字。为了让全球语系的用户可以彼此用计算机沟通，因此有了 Unicode 码。

Unicode 码的基本精神是，世上所有的文字都有一个码值，可以参考下列网页：

http://www.unicode.org/charts

目前，Unicode 内定义了超过 11 万的文字，它的定义方式是以“\u”开头后面有 4 个十六进制的数字，所以是从“\u0000”至“\uFFFF”。在上述网页中可以看到不同语系表，其中，East Asian Scripts 字段可以看到 CJK（Chinese，Japanese，Korean），在这里可以看到汉字的 Unicode 码值表，CJK 统一汉字的编码为 4E00 ～ 9FBB。

在 Unicode 编码中，前 128 个码值是保留给 ASCII 码使用，所以对于原先存在 ASCII 码中的英文大小写、标点符号等，是可以正常在 Unicode 码中使用的，Unicode 编码中经常用的是 ord( ) 函数。

ord (x)：可以返回函数字符参数 x 的 Unicode 码值，如果是中文字也可返回 Unicode 码值。如果是英文字符，Unicode 码值与 ASCII 码值是一样的。有了这个函数，可以很轻易地获得字符的 Unicode 码值。

程序实例 ch3_23.py：这个程序首先会将整数 97 转换成英文字符‘a’，然后将字符‘a’转换成 Unicode 码值，最后将中文字‘魁’转成 Unicode 码值。

```
1  # ch3_23.py
2  x1 = 97
3  x2 = chr(x1)
4  print(x2)                # 输出数值97的字符
5  x3 = ord(x2)
6  print(x3)                # 输出字符x3的Unicode(十进制)码值
7  x4 = '魁'
8  print(hex(ord(x4)))      # 输出字符'魁'的Unicode(十六进制)码值
```

执行结果



```
==================== RESTART: D:/Python/ch3/ch3_23.py ====================
a
97
0x9b41
```

3-5-3 utf-8 编码

utf-8 是针对 Unicode 字符集的可变长度编码方式，这是 Internet 目前所遵循的编码方式，在这种编码方式下，utf-8 使用 1 ～ 4 个 byte 表示一个字符，这种编码方式会根据不同的字符变化编码长度。

❑ ASCII 使用 utf-8 编码规则

对于 ASCII 字符而言，基本上它使用 1 个 byte 存储 ASCII 字符，utf-8 的编码方式是 byte 的第一个位是 0，其他 7 个位则是此字符的 ASCII 码值。

❑ 中文字的 utf-8 编码规则

对于需要 n 个 byte 编码的 Unicode 汉字字符而言，例如需要 3 个 byte 编码的汉字，第一个 byte 的前 n(3) 位皆设为 1，n+1(4) 设为 0。后面第 2 和第 3 个 byte 的前 2 位是 10，其他没有说明的二进制全部是此汉字字符的 Unicode 码。依照此规则，可以得到汉字的 utf-8 编码规则如下：

```
1110xxxx 10xxxxxx 10xxxxxx          # xx 就是要填入的 Unicode 码
```

例如，从 ch3_23.py 的执行结果可知“魁”的 Unicode 码值是 0x9b41，如果转成二进制方式则如下所示：

```
10011011 01000001
```

我们可以用下列更细的方式，将“魁”的 Unicode 码值填入 xx 内。

utf-8中文编码规则	1	1	1	0	x	x	x	x	1	0	x	x	x	x	x	x	1	0	x	x	x	x	x	x
魁的Unicode编码					1	0	0	1			1	0	1	1	0	1			0	0	0	0	0	1
魁的utf-8编码	1	1	1	0	1	0	0	1	1	0	1	0	1	1	0	1	1	0	0	0	0	0	0	1

从上图可以得到“魁”的 utf-8 编码结果是 0xe9ad81，3-6-1 节的实例 2 也可以验证这个结果。

3-6 bytes 数据

使用 Python 处理一般字符串数据时，可以很放心地使用 Unicode 字符串 str 数据类型，至于 Python 内部如何处理可以不用理会，这些事情 Python 的直译程序会处理。

但是有一天需要与外界沟通或交换数据时，特别是我们使用中文，如果不懂中文字符串与 bytes 数据的转换，所获得的数据将会是乱码。例如，设计电子邮件的接收程序，所接收的可能是 bytes 数据，这时必须学会将 bytes 数据转成 Unicode 字符串，否则会有乱码产生。或是有一天你要设计供中国人使用的网络聊天室，必须设计将使用者所传送的 Unicode 中文字符串转成 bytes 数据传上聊天室，然后也要设计将网络接收的 bytes 数据转成 Unicode 中文字符串，这个聊天室才可以顺畅使用。

bytes 数据格式是在字符串前加上 b，例如，下列是“魁”的 bytes 数据。

```
b'\xe9\xad\x81'
```

如果是英文字符串的 bytes 数据格式，相对单纯地会显示原始的字符，例如，下列是字符串“abc”的 bytes 数据。

```
b'abc'
```

3-6-1　Unicode 字符串转成 bytes 数据

将 Unicode 字符串转成 bytes 数据称为**编码**（encode），所使用的是 encode( ) 函数，这个方法的参数是指出编码的方法，可以参考下列表格。

编码	说明
'ascii'	标准 7 位的 ASCII 编码
'utf-8'	Unicode 可变长度编码，这也是最常使用的编码
'cp-1252'	一般英文 Windows 操作系统编码
'cp950'	繁体中文 Windows 操作系统编码
'unicode-escape'	Unicode 的常数格式，\uxxxx 或 \Uxxxxxxxx

如果 Unicode 字符串是英文则转成 bytes 数据相对容易，因为对于 utf-8 格式编码，Unicode 也是用一个 byte 存储每个字符串的字符。

实例 1：英文 Unicode 字符串数据转成 bytes 数据。

假设有一个字符串 string，内容是‘abc’，可以使用下列方法设置，同时检查此字符串的长度。

```
>>> string = 'abc'
>>> len(string)
3
```

下面将 Unicode 字符串 string 用 utf-8 编码格式转成 bytes 数据，然后列出 bytes 数据的长度、数据类型，以及 bytes 数据的内容。

```
>>> stringBytes = string.encode('utf-8')
>>> len(stringBytes)
3
>>> type(stringBytes)
<class 'bytes'>
>>> stringBytes
b'abc'
```

实例 2：中文 Unicode 字符串数据转成 bytes 数据。

假设有一个字符串 name，内容是‘洪锦魁’，可以使用下列方法设置，同时检查此字符串的长度。

```
>>> name = '洪锦魁'
>>> len(name)
3
```

下面将 Unicode 字符串 name 用 utf-8 编码格式转成 bytes 数据，然后列出 bytes 数据的长度、数据类型，以及 bytes 数据的内容。

```
>>> nameBytes = name.encode('utf-8')
>>> len(nameBytes)
9
>>> type(nameBytes)
<class 'bytes'>
>>> nameBytes
b'\xe6\xb4\xaa\xe9\x94\xa6\xe9\xad\x81'
```

由上述数据可以得到原来 Unicode 字符串用了 3byte 存储一个中文字，所以 3 个中文字获得了 bytes 的数据长度是 9。

3-6-2 bytes 数据转成 Unicode 字符串

对于一个专业的 Python 程序设计师而言，常常需要从网络取得数据，所取得的是 bytes 数据，这时需要将此数据转成 Unicode 字符串，将 bytes 数据转成 Unicode 字符串可以称为**译码**，所使用的是 decode( ) 函数，这个方法的参数是指出编码的方法，与 encode( ) 函数相同。

实例 1：bytes 数据转成 Unicode 字符串数据。

```
>>> stringUcode = stringBytes.decode('utf-8')
>>> len(stringUcode)
3
>>> stringUcode
'abc'
```

实例 2：bytes 数据转成 Unicode 字符串数据。

下面是将 nameBytes 数据使用 utf-8 编码格式转成 Unicode 字符串的方法，同时列出字符串长度和字符串内容。

```
>>> nameUcode = nameBytes.decode('utf-8')
>>> len(nameUcode)
3
>>> nameUcode
'洪锦魁'
```

3-7 专题——地球到月球时间计算 / 计算坐标轴两点之间的距离

3-7-1 计算地球到月球所需时间

马赫是音速的单位，主要是为了纪念奥地利科学家**恩斯特·马赫**（Ernst Mach）而命名，一马赫就是一倍音速，它的速度大约是每小时 1225 千米。

程序实例 ch3_24.py：从地球到月球约 384 400 千米，假设火箭的速度是一马赫，设计一个程序计算需要多少天、多少小时才可抵达月球。这个程序省略分钟数。

```
1  # ch3_24.py
2  dist = 384400                     # 地球到月亮距离
3  speed = 1225                      # 马赫速度每小时1225千米
4  total_hours = dist // speed       # 计算小时数
5  days = total_hours // 24          # 商 = 计算天数
6  hours = total_hours % 24          # 余数 = 计算小时数
7  print("总共需要天数")
8  print(days)
9  print("小时数")
10 print(hours)
```

执行结果

```
===================== RESTART: D:\Python\ch3\ch3_24.py =====================
总共需要天数
13
小时数
1
```

由于尚未介绍完整的格式化程序输出，所以使用上述方式输出，第 4 章会改良上述程序。Python 之所以可以成为当今最流行的程序语言，主要是它有丰富的函数库与方法，上述求商（第 5 行）和余数（第 6 行），在 2-9 节中介绍了 divmod( ) 函数，其实可以用 divmod( ) 函数一次取得商和余数，如下：

```
商 , 余数 = divmod( 被除数 , 除数 )              # 函数方法
days, hours = divmod(total_hours, 24)            # 本程序应用方式
```

程序实例 ch3_25.py：使用 divmod( ) 函数重新设计 ch3_24.py。

```
1  # ch3_25.py
2  dist = 384400                              # 地球到月亮距离
3  speed = 1225                               # 马赫速度每小时1225千米
4  total_hours = dist // speed                # 计算小时数
5  days, hours = divmod(total_hours, 24)      # 商和余数
6  print("总共需要天数")
7  print(days)
8  print("小时数")
9  print(hours)
```

执行结果 与 ch3_24.py 相同。

3-7-2　计算坐标轴两个点之间的距离

有两个点坐标分别是（x1, y1）、（x2, y2），这两个点的距离计算公式如下。

$$\sqrt{(x1-x2)^2+(y1-y2)^2}$$

可以将上述公式转成下列计算机数学表达式。

```
dist = ( (x1 - x2) ² + (y1 - y2) ² ) ** 0.5          # ** 0.5 相当于开根号
```

在人工智能的应用中，常用点坐标代表某一个对象的**特征**（feature），计算两个点之间的距离，相当于可以了解物体间的相似程度。距离越短代表相似度越高，距离越长代表相似度越低。

程序实例 ch3_26.py：有两个点坐标分别是 (1, 8）与 (3, 10)，请计算这两个点之间的距离。

```
1  # ch3_26.py
2  x1 = 1
3  y1 = 8
4  x2 = 3
5  y2 = 10
6  dist = ((x1 - x2) ** 2 + ((y1 - y2) ** 2)) ** 0.5
7  print("两点的距离是")
8  print(dist)
```

执行结果

```
==================== RESTART: D:\Python\ch3\ch3_26.py ====================
两点的距离是
2.8284271247461903
```

习题

1. 假设 a 是 10，b 是 18，c 是 5，请计算下列执行结果，取整数结果。(3-2 节)

(a) s = a + b – c　　(b) s = 2 * a + 3 – c　　(c) s = b * c + 20 / b

(d) s = a % c * b + 10　　(e) s = a ** c – a * b * c

```
===================== RESTART: D:\Python\ex\ex3_1.py =====================
13
18
42.5
10
99600
```

2. 请重新设计第 2 章习题 2，请使用 int( ) 函数，以整数列出本金和。(3-2 节)

```
===================== RESTART: D:\Python\ex\ex3_2.py =====================
本金和
106120
```

3. 请重新设计第 2 章习题 2，使用 round( ) 函数，以整数列出本金和。(3-2 节)

```
====================== RESTART: D:\Python\ex\ex3_3.py ======================
本金和
106121
```

4. 地球和月球的距离是 384 400 千米，假设火箭飞行速度是每分钟 250 千米，请问从地球飞到月球需要多少天、多少小时、多少分钟，请舍去秒钟。(3-2 节)

```
====================== RESTART: D:/Python/ex/ex3_4.py ======================
天总数
1
小时数
25
分钟数
37
```

5. 请列出你自己名字十进制的 Unicode 码值。(3-5 节)

```
====================== RESTART: D:/Python/ex/ex3_5.py ======================
洪
27946
锦
38182
魁
39745
```

6. 请列出你自己名字十六进制的 Unicode 码值。(3-5 节)

```
======================= RESTART: D:/Python/ex/ex3_6.py =======================
洪
0x6d2a
锦
0x9526
魁
0x9b41
```

7. 请将 Unicode 字符串“Python 王者归来”转成 bytes 数据，然后输出 bytes 数据。(3-6 节)

```
======================= RESTART: D:/Python/ex/ex3_7.py =======================
Unicode字符串内容
Python王者归来
bytes数据内容
b'Python\xe7\x8e\x8b\xe8\x80\x85\xe5\xbd\x92\xe6\x9d\xa5'
将bytes数据转回Unicode字符串
Python王者归来
```

8. 重新设计 ch3_25.py，需计算至分钟与秒钟数。(3-7 节)

```
======================= RESTART: D:/Python/ex/ex3_8.py =======================
313.7959183673469
总共需要天数
13.0
小时数
1.7959183673469283
分钟数
47
秒钟数
45
```

9. 请修改 ch3_26.py，计算这两个点坐标（1, 8）与（3, 10）距坐标原点（0, 0）的距离。

```
======================= RESTART: D:/Python/ex/ex3_9.py =======================
坐标(1, 8)点与坐标原点(0, 0)的距离是
8.06225774829855
坐标(3, 10)点与坐标原点(0, 0)的距离是
10.44030650891055
```

04

第 4 章

基本输入与输出

本章摘要

本章将介绍如何在屏幕上进行输入与输出，另外也将讲解 Python 内建的实用功能。

4-1 Python 的辅助说明 help( )

help( ) 函数可以列出某一个 Python 指令或函数的使用说明。

实例：列出输出函数 print( ) 的使用说明。

```
>>> help(print)
Help on built-in function print in module builtins:

print(...)
    print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

    Prints the values to a stream, or to sys.stdout by default.
    Optional keyword arguments:
    file:  a file-like object (stream); defaults to the current sys.stdout.
    sep:   string inserted between values, default a space.
    end:   string appended after the last value, default a newline.
    flush: whether to forcibly flush the stream.

>>>
```

当然程序语言是全球化的语言，所有说明是以英文为基础，要有一定的英文能力才可彻底了解，不过笔者在本书中会详尽地用中文引导读者入门。

4-2 格式化输出数据使用 print( )

相信读者经过前三章的学习，对于使用 print( ) 函数输出数据已经非常熟悉了，现在是时候完整解说这个输出函数的用法了。

4-2-1　函数 print( ) 的基本语法

print( ) 的基本语法格式如下：

```
print(value, … , sep=" ", end="\n", file=sys.stdout, flush=False)
```

value：表示想要输出的数据，可以一次输出多个数据，各数据间以逗号隔开。

sep：当输出多个数据时，可以插入各数据的分隔字符，默认是一个空格。

end：当数据输出结束时所插入的字符，默认是插入换行字符，所以下一次 print( ) 函数的输出会在下一行输出。如果想让下次输出不换行，可以在此设置空字符串，或是空格或是其他字符串。

file：数据输出位置，默认是 sys.stdout，也就是屏幕。也可以使用此设置，将输出导入其他文件或设备。

flush：是否清除数据流的缓冲区，默认是不清除。

程序实例 ch4_1.py：重新设计 ch3_18.py，其中在第二个 print( ) 中两个输出数据的分隔字符是“ $$$”。

```
1  # ch4_1.py
2  num1 = 222
3  num2 = 333
4  num3 = num1 + num2
5  print("这是数值相加", num3)
6  str1 = str(num1) + str(num2)
7  print("强制转换为字符串相加", str1, sep=" $$$ ")
```

执行结果

```
===================== RESTART: D:\Python\ch4\ch4_1.py =====================
这是数值相加 555
强制转换为字符串相加 $$$ 222333
```

程序实例 ch4_2.py：重新设计 ch4_1.py，将两个数据在同一行输出，彼此之间使用 Tab 键的距离隔开。

```
1  # ch4_2.py
2  num1 = 222
3  num2 = 333
4  num3 = num1 + num2
5  print("这是数值相加", num3, end="\t")  # 以Tab键值位置分隔两个数据输出
6  str1 = str(num1) + str(num2)
7  print("强制转换为字符串相加", str1, sep=" $$$ ")
```

执行结果

```
===================== RESTART: D:\Python\ch4\ch4_2.py =====================
这是数值相加 555        强制转换为字符串相加 $$$ 222333
```

4-2-2　格式化 print( ) 输出

在使用格式化输出时，基本使用格式如下：

print(" …输出格式区… " % (变量系列区， …))

在上述**输出格式区**中，可以放置**变量系列区**相对应的格式化字符，这些格式化字符的基本意义如下。

%d：格式化整数输出。

%f：格式化浮点数输出。

%x：格式化十六进制整数输出。

%X：格式化大写十六进制整数输出。

%o：格式化八进制整数输出。

%s：格式化字符串输出。

%e：格式化科学记数法 e 的输出。

%E：格式化科学记数法大写 E 的输出。

程序实例 ch4_3.py：格式化输出的应用。

```
1  # ch4_3.py
2  score = 90
3  name = "洪锦魁"
4  count = 1
5  print("%s你的第 %d 次物理考试成绩是 %d" % (name, count, score))
```

执行结果

```
======================= RESTART: D:\Python\ch4\ch4_3.py =======================
洪锦魁你的第 1 次物理考试成绩是 90
```

设计程序时，在 print( ) 函数内的**输出格式区**也可以用一个字符串变量取代。

程序实例 ch4_4.py：重新设计 ch4_3.py，在 print( ) 内用字符串变量取代字符串列，读者可以参考第 5 行和第 6 行与原先 ch4_3.py 的第 5 行做比较。

```
1  # ch4_4.py
2  score = 90
3  name = "洪锦魁"
4  count = 1
5  formatstr = "%s你的第 %d 次物理考试成绩是 %d"
6  print(formatstr % (name, count, score))
```

执行结果 与 ch4_3.py 相同。

程序实例 ch4_5.py：格式化十六进制和八进制输出的应用。

```
1  # ch4_5.py
2  x = 100
3  print("100的十六进制 = %x\n100的八进制 = %o" % (x, x))
```

执行结果

```
======================= RESTART: D:\Python\ch4\ch4_5.py =======================
100的十六进制 = 64
100的八进制 = 144
```

程序实例 ch4_6.py：将整数与浮点数分别以 %d、%f、%s 格式化，同时观察执行结果。特别要注意的是，浮点数以整数 %d 格式化后，小数数据将被舍去。

```
1  # ch4_6.py
2  x = 10
3  print("整数%d \n浮点数%f \n字符串%s" % (x, x, x))
4  y = 9.9
5  print("整数%d \n浮点数%f \n字符串%s" % (y, y, y))
```

执行结果

```
======================= RESTART: D:\Python\ch4\ch4_6.py =======================
整数10
浮点数10.000000
字符串10
整数9
浮点数9.900000
字符串9.9
```

下列是使用 %x 和 %X 格式化数据输出的实例。

```
>>> x = 27
>>> print("%x" % x)
1b
>>> print("%X" % x)
1B
```

下列是使用 %e 和 %E 格式化科学记数法数据输出的实例。

```
>>> x = 10000000
>>> print("%e" % x)
1.000000e+07
>>> print("%E" % x)
1.000000E+07
>>> y = 0.000123
>>> print("%e" % y)
1.230000e-04
```

4-2-3 精准控制格式化的输出

在上述程序实例 ch4_6.py 中，我们发现最大的缺点是无法精确地控制浮点数的小数输出位数，print( ) 函数在格式化过程中，可以让我们设置保留多少格的空间让文件做输出，此时格式化的语法如下。

%（+|-）nd：格式化整数输出。

%（+|-）m.nf：格式化浮点数输出。

%（+|-）nx：格式化十六进制整数输出。

%（+|-）no：格式化八进制整数输出。

%（-）ns：格式化字符串输出。

%（-）m.ns：m 是输出字符串宽度，n 是显示字符串长度，n 小于字符串长度时会有裁减字符串的效果。

%（+|-）e：格式化科学记数法 e 输出。

%（+|-）E：格式化科学记数法大写 E 输出。

上述格式对浮点数而言，m 代表保留多少格数供输出（包含小数点），n 则是小数数据保留格数。至于其他的数据格式，n 则是保留多少格数空间，如果保留格数空间不足将完整输出数据，如果保留格数空间太多则数据靠右对齐。

如果是格式化数值数据或字符串数据有加上负号（-），表示保留格数空间有多余时，数据将靠左输出。如果是格式化数值数据有加上正号（+），如果输出数据是正值时，将在左边加上正值符号。

程序实例 ch4_7.py：格式化输出的应用。

```
1  # ch4_7.py
2  x = 100
3  print("x=/%6d/" % x)
4  y = 10.5
5  print("y=/%6.2f/" % y)
6  s = "Deep"
7  print("s=/%6s/" % s)
8  print("以下是保留格数空间不足的实例")
9  print("x=/%2d/" % x)
10 print("y=/%3.2f/" % y)
11 print("s=/%2s/" % s)
```

执行结果

```
====================== RESTART: D:\Python\ch4\ch4_7.py ======================
x=/   100/
y=/ 10.50/
s=/  Deep/
以下是保留格数空间不足的实例
x=/100/
y=/10.50/
s=/Deep/
```

程序实例 ch4_8.py：格式化输出，靠左对齐的实例。

```
1  # ch4_8.py
2  x = 100
3  print("x=/%-6d/" % x)
4  y = 10.5
5  print("y=/%-6.2f/" % y)
6  s = "Deep"
7  print("s=/%-6s/" % s)
```

执行结果

```
====================== RESTART: D:/Python/ch4/ch4_8.py ======================
x=/100   /
y=/10.50 /
s=/Deep  /
```

程序实例 ch4_9.py：格式化输出，正值数据将出现正号（+）。

```
1  # ch4_9.py
2  x = 10
3  print("x=/%+6d/" % x)
4  y = 10.5
5  print("y=/%+6.2f/" % y)
```

执行结果

```
====================== RESTART: D:/Python/ch4/ch4_9.py ======================
x=/   +10/
y=/+10.50/
```

程序实例 ch4_10.py：格式化输出的应用。

```
1  # ch4_10.py
2  print(" 姓名    语文    英文    总分")
3  print("%3s  %4d    %4d    %4d" % ("洪冰儒", 98, 90, 188))
4  print("%3s  %4d    %4d    %4d" % ("洪雨星", 96, 95, 191))
5  print("%3s  %4d    %4d    %4d" % ("洪冰雨", 92, 88, 180))
6  print("%3s  %4d    %4d    %4d" % ("洪星宇", 93, 97, 190))
```

执行结果

```
====================== RESTART: D:\Python\ch4\ch4_10.py ======================
 姓名    语文    英文    总分
洪冰儒    98     90     188
洪雨星    96     95     191
洪冰雨    92     88     180
洪星宇    93     97     190
```

下面是格式化科学记数法 e 和 E 输出的应用。

```
>>> x = 12345678
>>> print("/%10.1e/" % x)
/   1.2e+07/
>>> print("/%10.2E/" % x)
/  1.23E+07/
>>> print("/%-10.2E/" % x)
/1.23E+07  /
>>> print("/%+10.2E/" % x)
/ +1.23E+07/
```

对于格式化字符串有一个特别的是使用“%m.n”方式格式化字符串，这时 m 是保留显示字符串空间，n 是显示字符串长度，如果 n 的长度小于实际字符串长度，会有裁减字符串的效果。

```
>>> string = "abcdefg"
>>> print("/%10.3s/" % string)
/       abc/
```

4-2-4　format( ) 函数

这是 Python 增强版的格式化输出功能，是字符串使用 format 方法做格式化的动作，基本使用格式如下：

print("　…输出格式区…　" .format(变量系列区 ,　…　))

在输出格式区内的变量使用“{ }”表示。

程序实例 ch4_11.py：使用 format( ) 函数重新设计 ch4_3.py。

```
1  # ch4_11.py
2  score = 90
3  name = "洪锦魁"
4  count = 1
5  print("{}你的第 {} 次物理考试成绩是 {}".format(name, count, score))
```

执行结果 与 ch4_3.py 相同。

程序实例 ch4_12.py：以字符串代表输出格式区，重新设计 ch4_11.py。

```
1  # ch4_12.py
2  score = 90
3  name = "洪锦魁"
4  count = 1
5  string = "{}你的第 {} 次物理考试成绩是 {}"
6  print(string.format(name, count, score))
```

执行结果 与 ch4_3.py 相同。

在使用 { } 代表变量时，也可以在 { } 内增加编号 n，此时 n 将是 format( ) 内变量的顺序，编号从 0 开始计算，变量多时方便了解变量的顺序。

程序实例 ch4_12_1.py：重新设计 ch4_12.py，在 { } 内增加编号。

```
1  # ch4_12_1.py
2  score = 90
3  name = "洪锦魁"
4  count = 1
5  # 以下鼓励使用
6  print("{0}你的第 {1} 次物理考试成绩是 {2}".format(name,count,score))
7
8  # 以下语法对但不鼓励使用
9  print("{2}你的第 {1} 次物理考试成绩是 {0}".format(score,count,name))
```

执行结果

```
===================== RESTART: D:\Python\ch4\ch4_12_1.py =====================
洪锦魁你的第 1 次物理考试成绩是 90
洪锦魁你的第 1 次物理考试成绩是 90
```

也可以在 format( ) 内使用具名参数。

程序实例 ch4_12_2.py：使用具名参数，重新设计 ch4_12_1.py。

```
1  # ch4_12_2.py
2  print("{n}你的第 {c} 次物理考试成绩是 {s}".format(n="洪锦魁",c=1,s=90))
```

执行结果

```
===================== RESTART: D:\Python\ch4\ch4_12_2.py =====================
洪锦魁你的第 1 次物理考试成绩是 90
```

也可以将 4-2-2 节所述格式化输出数据的概念应用于 format( )，例如，d 是格式化整数、f 是格式化浮点数、s 是格式化字符串等。传统的格式化输出是使用 % 配合 d、s、f，使用 format 则是使用“:”，可参考下列实例第 5 行。

程序实例 ch4_12_3.py：计算圆面积，同时格式化输出。

```
1  # ch4_12_3.py
2  r = 5
3  PI = 3.14159
4  area = PI * r ** 2
5  print("/半径{0:3d}圆面积是{1:10.2f}/".format(r,area))
```

执行结果

```
===================== RESTART: D:\Python\ch4\ch4_12_3.py =====================
/半径  5圆面积是     78.54/
```

在使用格式化输出时默认是靠右输出，也可以使用下列参数设置输出对齐方式。

\>：靠右对齐

<：靠左对齐

^：居中对齐

程序实例 ch4_12_4.py：输出对齐方式的应用。

```
1  # ch4_12_4.py
2  r = 5
3  PI = 3.14159
4  area = PI * r ** 2
5  print("/半径{0:3d}圆面积是{1:10.2f}/".format(r,area))
6  print("/半径{0:>3d}圆面积是{1:>10.2f}/".format(r,area))
7  print("/半径{0:<3d}圆面积是{1:<10.2f}/".format(r,area))
8  print("/半径{0:^3d}圆面积是{1:^10.2f}/".format(r,area))
```

执行结果

```
===================== RESTART: D:\Python\ch4\ch4_12_4.py =====================
/半径  5圆面积是     78.54/
/半径  5圆面积是     78.54/
/半径5  圆面积是78.54     /
/半径 5 圆面积是  78.54   /
```

在使用 format 输出时也可以使用填充字符，字符是放在“：”后面，在 <、^、> 或指定宽度之前。

程序实例 ch4_12_5.py：填充字符的应用。

```
1  # ch4_12_5.py
2  title = "南极旅游讲座"
3  print("/{0:*^20s}/".format(title))
```

执行结果

```
===================== RESTART: D:\Python\ch4\ch4_12_5.py =====================
/*******南极旅游讲座*******/
```

4-2-5　字符串输出与基本排版的应用

适度利用输出格式，可以制作一封排版的信件，以下程序的前 3 行会先利用 sp 字符串变量建立一个含 40 格的空白格数，然后产生对齐效果。

程序实例 ch4_12_6.py：有趣排版信件的应用。

```
 1  # ch4_12_6.py
 2  sp = " " * 40
 3  print("%s    1231 Delta Rd" % sp)
 4  print("%s    Oxford, Mississippi" % sp)
 5  print("%s    USA\n\n\n" % sp)
 6  print("Dear Ivan")
 7  print("I am pleased to inform you that your application for fall 2020 has")
 8  print("been favorably reviewed by the Electrical and Computer Engineering")
 9  print("Office.\n\n")
10  print("Best Regards")
11  print("Peter Malong")
```

执行结果

```
===================== RESTART: D:\Python\ch4\ch4_12_6.py =====================
                                            1231 Delta Rd
                                            Oxford, Mississippi
                                            USA



Dear Ivan
I am pleased to inform you that your application for fall 2020 has
been favorably reviewed by the Electrical and Computer Engineering
Office.


Best Regards
Peter Malong
```

4-2-6　一个无聊的操作

程序实例 ch4_12_6.py 第 2 行，利用空格乘以 40 产生 40 个空格，功能是用于排版。如果将某个字符串乘以 500，然后用 print( ) 输出，可以在屏幕上建立一个无聊的画面。

实例：在屏幕上建立一个无聊的画面。

```
>>> x = "Boring Time" * 500
>>> print(x)
Boring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBorin
g TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring Tim
eBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBori
ng TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring Ti
meBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBor
ing TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring T
imeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBo
ring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring
TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeB
oring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring
TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeB
oring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring
TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeB
oring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring
```

上述实例是启发读者活用 Python，可以产生许多意外的结果。

4-3　输出数据到文件

在 4-2-1 节有讲解在 print( ) 函数中，默认输出位置是屏幕（sys.stdout），其实可以利用这个特性将输出导向一个文件。

4-3-1　打开一个文件 open( )

open( ) 函数可以打开一个文件供读取或写入，如果这个函数执行成功，会返回文件流对象，这个函数的基本使用格式如下：

```
file_Obj = open(file, mode="r" )# 只列出最常用的两个参数
```

file：用字符串列出要打开的文件，如果不指明路径，则打开目前工作文件夹。

mode：打开文件的模式，如果省略代表是 mode="r"，使用时如果 mode="w" 或其他，也可以省略“mode=”，直接写“w”。也可以同时具有多项模式，例如，“wb”代表以二进制文件打开供写入，可以是下列基本模式。下列是第一个字母的操作意义。

- "r"：这是默认值，打开文件供读取（read）。
- "w"：打开文件供写入，如果原先文件有内容将被覆盖。
- "a"：打开文件供写入，如果原先文件有内容，新写入数据将附加在后面。
- "x"：打开一个新的文件供写入，如果所打开的文件已经存在会产生错误。

下列是第二个字母的意义，代表文件类型。

- "b"：打开二进制文件模式。
- "t"：打开文本文件模式，这是默认值。

file_Obj：这是文件对象，读者可以自行命名，未来 print( ) 函数可以将输出导向此对象，不使用时要关闭 file_Obj.close( )，才可以返回操作系统的文件管理器观察执行结果。

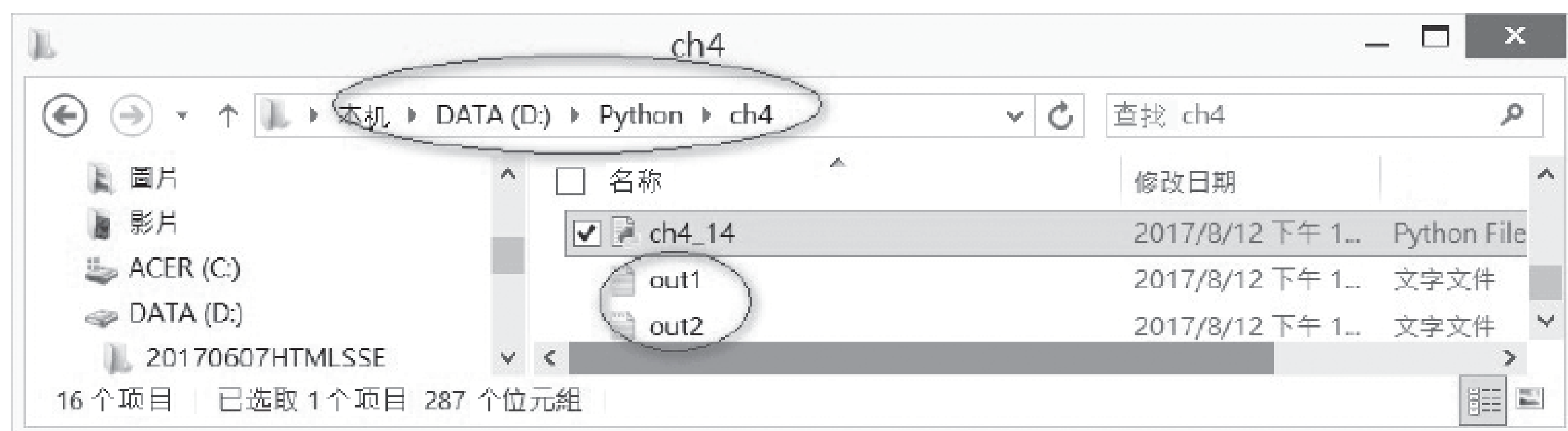
4-3-2 使用 print( ) 函数输出数据到文件

程序实例 ch4_13.py：将数据输出到文件的实例，其中，输出到 out1.txt 采用“w”模式，输出到 out2.txt 采用“a”模式。

```
1  # ch4_13.py
2  fstream1 = open("d:\python\ch4\out1.txt", mode="w") # 覆盖先前文件
3  print("Testing for output", file=fstream1)
4  fstream1.close()
5  fstream2 = open("d:\python\ch4\out2.txt", mode="a") # 附加数据后面
6  print("Testing for output", file=fstream2)
7  fstream2.close()
```

执行结果

这个程序执行后需到 ch4 文件夹查看执行结果内容，如果执行程序一次，可以得到 out1.txt 和 out2.txt 内容相同。但是如果持续执行，out2.txt 内容会持续增加，out1.txt 内容则保持不变，下列是检查文件夹内容。



下列是执行两次此程序后 out1.txt 和 out2.txt 的内容。



4-4 数据输入 input( )

这个 input( ) 函数功能与 print( ) 函数功能相反，会从屏幕读取用户从键盘输入的数据，它的使用格式如下：

```
value = input("prompt: ")
```

value 是变量，所输入的数据会存储在此变量内，特别需注意的是所输入的数据不论是字符串或是数值数据返回到 value 时一律是字符串数据，如果要执行数学运算需要用 int( ) 函数转换为整数。

程序实例 ch4_14.py：认识输入数据类型。

```
1  # ch4_14.py
2  name = input("请输入姓名：")
3  engh = input("请输入成绩：")
4  print("name数据类型是", type(name))
5  print("engh数据类型是", type(engh))
```

执行结果

```
====================== RESTART: D:\Python\ch4\ch4_14.py ======================
请输入姓名：洪锦魁
请输入成绩：100
name数据类型是 <class 'str'>
engh数据类型是 <class 'str'>
```

程序实例 ch4_15.py：基本数据输入与运算。

```
1  # ch4_15.py
2  print("欢迎使用成绩输入系统")
3  name = input("请输入姓名：")
4  engh = input("请输入英文成绩：")
5  math = input("请输入数学成绩：")
6  total = int(engh) + int(math)
7  print("%s 你的总分是 %d" % (name, total))
```

执行结果

```
====================== RESTART: D:\Python\ch4\ch4_15.py ======================
欢迎使用成绩输入系统
请输入姓名：洪锦魁
请输入英文成绩：98
请输入数学成绩：99
洪锦魁 你的总分是 197
```

接下来的程序主要是处理中文名字与英文名字的技巧，假设要求使用者分别输入**姓氏**（lastname）与**名字**（firstname），在中文中要处理成名字，可以使用下列字符串连接方式。

```
fullname = lastname + firstname
```

在英文中首先**名字在前面，姓氏在后面**，同时中间有一个空格，因此处理方式如下：

```
fullname = firstname + " " + lastname
```

程序实例 ch4_16.py：分别输入中文和英文的姓氏以及名字，本程序将会输出名字组合并输出问候语。

```
1  # ch4_16.py
2  clastname = input("请输入中文姓氏：")
3  cfirstname = input("请输入中文名字：")
4  cfullname = clastname + cfirstname
5  print("%s 欢迎使用本系统" % cfullname)
6  lastname = input("请输入英文Last Name：")
7  firstname = input("请输入英文First Name：")
8  fullname = firstname + " " + lastname
9  print("%s Welcome to SSE System" % fullname)
```

执行结果

```
====================== RESTART: D:\Python\ch4\ch4_16.py ======================
请输入中文姓氏：洪
请输入中文名字：锦魁
洪锦魁 欢迎使用本系统
请输入英文Last Name：Hung
请输入英文First Name：Jiin-Kwei
Jiin-Kwei Hung Welcome to SSE System
```

4-5 处理字符串的数学运算 eval( )

Python 内有一个非常好用的计算数学表达式的函数 eval( )，这个函数可以直接返回字符串内数学表达式的计算结果。

```
result = eval(expression)          # expression 是字符串
```

程序实例 ch4_17.py：输入公式，本程序可以列出计算结果。

```
1  # ch4_17.py
2  numberStr = input("请输入数值公式 : ")
3  number = eval(numberStr)
4  print("计算结果 : %5.2f" % number)
```

执行结果

```
===================== RESTART: D:\Python\ch4\ch4_17.py =====================
请输入数值公式 : 5*9+10
计算结果 : 55.00
>>>
===================== RESTART: D:\Python\ch4\ch4_17.py =====================
请输入数值公式 : 5 * 9 + 10
计算结果 : 55.00
```

由上述执行结果可以发现，在第一个执行结果中输入的是“5*9+10”字符串，eval( ) 函数可以处理此字符串的数学表达式，然后将计算结果返回，同时也可以发现即使此数学表达式之间有空字符也可以正常处理。

Windows 操作系统有计算器程序，其实当我们使用计算器输入运算公式时，就可以将所输入的公式用字符串存储，然后使用 eval( ) 方法就可以得到运算结果。在 ch4_15.py 中 input( ) 所输入的数据是字符串，当时我们使用 int( ) 将字符串转成整数处理，其实也可以使用 eval( ) 配合 input( )，直接返回整数数据。

程序实例 ch4_18.py：使用 eval( ) 重新设计 ch4_15.py。

```
1  # ch4_18.py
2  print("欢迎使用成绩输入系统")
3  name = input("请输入姓名：")
4  engh = eval(input("请输入英文成绩："))
5  math = eval(input("请输入数学成绩："))
6  total = engh + math
7  print("%s 你的总分是 %d" % (name, total))
```

执行结果

```
===================== RESTART: D:\Python\ch4\ch4_18.py =====================
欢迎使用成绩输入系统
请输入姓名：洪锦魁
请输入英文成绩：98
请输入数学成绩：99
洪锦魁 你的总分是 197
```

一个 input( ) 可以读取一个输入字符串，我们可以灵活运用多重指定在 eval( ) 与 input( ) 函数上，然后产生一行输入多个数值数据的效果。

程序实例 ch4_19.py：输入 3 个数字，本程序可以输出平均值，注意输入时各数字间要用“,”隔开。

```
1  # ch4_19.py
2  n1, n2, n3 = eval(input("请输入3个数字："))
3  average = (n1 + n2 + n3) / 3
4  print("3个数字平均是 %6.2f" % average)
```

执行结果

```
====================== RESTART: D:\Python\ch4\ch4_19.py ======================
请输入3个数字：21, 33, 99
3个数字平均是  51.00
```

4-6 列出所有内建函数 dir( )

阅读至此，相信读者已经使用了许多 Python 内建的函数了，例如 help( )、print( )、input( ) 等，读者可能想了解到底 Python 提供哪些内建函数可供我们在设计程序时使用，可以使用下列方式列出 Python 所提供的内建函数。

```
dir(__builtins__)    # 列出 Python 内建函数
```

实例：列出 Python 所有内建函数。

```
>>> dir(__builtins__)
['ArithmeticError', 'AssertionError', 'AttributeError', 'BaseException', 'BlockingIOError', 'BrokenPipeE
rror', 'BufferError', 'BytesWarning', 'ChildProcessError', 'ConnectionAbortedError', 'ConnectionError',
'ConnectionRefusedError', 'ConnectionResetError', 'DeprecationWarning', 'EOFError', 'Ellipsis', 'Environ
mentError', 'Exception', 'False', 'FileExistsError', 'FileNotFoundError', 'FloatingPointError', 'FutureW
arning', 'GeneratorExit', 'IOError', 'ImportError', 'ImportWarning', 'IndentationError', 'IndexError', '
InterruptedError', 'IsADirectoryError', 'KeyError', 'KeyboardInterrupt', 'LookupError', 'MemoryError', '
ModuleNotFoundError', 'NameError', 'None', 'NotADirectoryError', 'NotImplemented', 'NotImplementedError'
, 'OSError', 'OverflowError', 'PendingDeprecationWarning', 'PermissionError', 'ProcessLookupError', 'Rec
ursionError', 'ReferenceError', 'ResourceWarning', 'RuntimeError', 'RuntimeWarning', 'StopAsyncIteration
', 'StopIteration', 'SyntaxError', 'SyntaxWarning', 'SystemError', 'SystemExit', 'TabError', 'TimeoutErr
or', 'True', 'TypeError', 'UnboundLocalError', 'UnicodeDecodeError', 'UnicodeEncodeError', 'UnicodeError
', 'UnicodeTranslateError', 'UnicodeWarning', 'UserWarning', 'ValueError', 'Warning', 'WindowsError', 'Z
eroDivisionError', '__build_class__', '__debug__', '__doc__', '__import__', '__loader__', '__name__', '_
_package__', '__spec__', 'abs', 'all', 'any', 'ascii', 'bin', 'bool', 'bytearray', 'bytes', 'callable',
'chr', 'classmethod', 'compile', 'complex', 'copyright', 'credits', 'delattr', 'dict', 'dir', 'divmod',
'enumerate', 'eval', 'exec', 'exit', 'filter', 'float', 'format', 'frozenset', 'getattr', 'globals', 'ha
sattr', 'hash', 'help', 'hex', 'id', 'input', 'int', 'isinstance', 'issubclass', 'iter', 'len', 'license
', 'list', 'locals', 'map', 'max', 'memoryview', 'min', 'next', 'object', 'oct', 'open', 'ord', 'pow', '
print', 'property', 'quit', 'range', 'repr', 'reversed', 'round', 'set', 'setattr', 'slice', 'sorted', '
staticmethod', 'str', 'sum', 'super', 'tuple', 'type', 'vars', 'zip']
>>>
```

在本书中，笔者会依功能分类将常用的内建函数分别融入各章节主题中，如果读者想了解某一个内建函数的功能，可参考 4-1 节使用 help( ) 函数。

4-7 专题——温度转换 / 房贷问题 / 正五角形面积 / 利用经纬度计算距离

4-7-1 设计摄氏温度和华氏温度的转换

摄氏温度（Celsius，C）的由来是在标准大气压环境，纯水的凝固点是 0℃，沸点是 100℃，中间划分 100 等份，每个等份是摄氏 1 度。为了纪念瑞典科学家**安德斯·摄尔修斯**（Anders Celsius）

对摄氏温度定义的贡献，所以称为**摄氏温度**（Celsius）。

华氏温度（Fahrenheit，F）的由来是在标准大气压环境，水的凝固点是 32℃、水的沸点是 212℃，中间划分 180 等份，每个等份是华氏 1 度。为了纪念德国科学家**丹尼尔·加布里埃尔·华伦海特**（Daniel Gabriel Fahrenheit）对华氏温度定义的贡献，所以称为**华氏温度**（Fahrenheit）。

摄氏和华氏温度互转的公式如下：

$$\text{摄氏温度} = (\text{华氏温度} - 32) \times 5 / 9$$

$$\text{华氏温度} = \text{摄氏温度} \times (9 / 5) + 32$$

程序实例 ch4_20.py：请输入华氏温度，这个程序会输出摄氏温度。

```
1  # ch4_20.py
2  f = input("请输入华氏温度：")
3  c = ( int(f) - 32 ) * 5 / 9
4  print("华氏 %s 等于摄氏 %4.1f" % (f, c))
```

执行结果

```
====================== RESTART: D:\Python\ch4\ch4_20.py ======================
请输入华氏温度：104
华氏 104 等于摄氏 40.0
>>>
====================== RESTART: D:\Python\ch4\ch4_20.py ======================
请输入华氏温度：88
华氏 88 等于摄氏 31.1
```

4-7-2 房屋贷款问题

每个人在成长的过程中可能都会经历买房子，第一次住在属于自己的房子中是一个美好的经历，大多数人在这个过程中可能需要向银行贷款。这时会思考需要贷多少钱？贷款年限是多少？银行利率是多少？然后可以利用上述已知资料计算每个月还款金额是多少，同时我们会好奇整个贷款结束究竟还了多少贷款本金和利息。在做这个专题分析时，已知的条件是：

贷款金额：使用 loan 当变量

贷款年限：使用 year 当变量

年利率：使用 rate 当变量

然后需要利用上述条件计算下列结果。

每月还款金额：使用 monthlyPay 当变量

总共还款金额：使用 totalPay 当变量

处理这个贷款问题的数学公式如下：

$$\text{每月还款金额} = \frac{\text{贷款金额} \times \text{月利率}}{1 - \dfrac{1}{(1+\text{月利率})^{\text{贷款年限} \times 12}}}$$

在银行的贷款术语习惯使用年利率，所以碰上这类问题需要将所输入的利率先除以 100，这是转成百分比，同时要除以 12 表示是月利率。可以用下列方式计算月利率，用 monthrate 当变量。

```
monthrate = rate / (12*100)                    # 第 5 行
```

为了不让求每月还款金额的数学式变得复杂，将分子（第 8 行）与分母（第 9 行）分开计算，第 10 行是计算每月还款金额，第 11 行是计算总共还款金额。

程序实例 ch4_21.py：请输入贷款金额、贷款年限和年利率，程序会输出每月还款金额和总共还款金额。

```
1  # ch4_21.py
2  loan = eval(input("请输入贷款金额："))
3  year = eval(input("请输入年限："))
4  rate = eval(input("请输入年利率："))
5  monthrate = rate / (12*100)            # 改成百分比以及月利率
6
7  # 计算每月还款金额
8  molecules = loan * monthrate
9  denominator = 1 - (1 / (1 + monthrate) ** (year * 12))
10 monthlyPay = molecules / denominator    # 每月还款金额
11 totalPay = monthlyPay * year * 12       # 总共还款金额
12
13 print("每月还款金额 %d" % int(monthlyPay))
14 print("总共还款金额 %d" % int(totalPay))
```

执行结果

```
====================== RESTART: D:\Python\ch4\ch4_21.py ======================
请输入贷款金额：6000000
请输入年限：20
请输入年利率：2.0
每月还款金额 30353
总共还款金额 7284720
```

4-7-3　正五角形面积

在几何学中正五角形边长假设是 s，其面积的计算公式如下：

$$\textbf{area} = \frac{5 \times s^2}{4 \times \tan\left(\frac{\pi}{5}\right)}$$

上述计算正五角形面积需要使用数学中的 PI，虽然可以使用 3.14159 代替，不过笔者此处先引导读者学习使用 Python 的数学模块，有关模块的概念将在第 13 章说明，此节将先教导读者使用，可以使用“import math”导入此数学模块。

程序实例 ch4_22.py：请输入正五角形的边长 s，此程序会计算此正五角形的面积。

```
1  # ch4_22.py
2  import math
3
4  s = eval(input("请输入正五角形边长 ： "))
5  area = (5 * s ** 2) / (4 * math.tan(math.pi / 5))
6  print("area = ", area)
```

执行结果

```
====================== RESTART: D:\Python\ch4\ch4_22.py ======================
请输入正五角形边长 ： 5
area =  43.01193501472417
```

可以将上述概念扩充应用在正多边形面积计算，相关概念可以参考习题 13。

4-7-4 利用经纬度计算地球各城市间的距离

地球是圆的，我们可以使用经度和纬度来了解地球上每一个点的位置。有了两个地点的经纬度后，可以使用下列公式计算彼此的距离。

distance = r×acos(sin(x1)×sin(x2)+cos(x1)×cos(x2)×cos(y1-y2))

上述 r 是地球的半径约 6371 千米，由于 Python 的三角函数都是弧度（radians）单位，我们使用上述公式时，需使用 math.radian( ) 函数将角度转成弧度。上述公式西经和北纬是正值，东经和南纬是负值。

经度坐标介于 -180° ～ 180° ，纬度坐标是 -90° ～和 90° ，虽然我们习惯称经纬度，在用小括号表达时却是（纬度 , 经度），也就是第一个参数放纬度，第二个参数放经度。

最简单的获得经纬度的方式是打开 Google 地图，其实打开 Google 地图后就可以在网址列看到我们目前所在地点的经纬度，选择地点就可以在网址列看到所选地点的经纬度信息，可参考下方左图。



由上图可以知道中国台北车站的经纬度是（25.0452909, 121.5168704），以上概念可以应用于查询世界各地的经纬度，上方右图是中国香港红磡车站的经纬度（22.2838912, 114.173166），程序为了简化小数取 4 位。

程序实例 ch4_23.py：中国香港红磡车站的经纬度信息是（22.2839, 114.1731），中国台北车站的经纬度是（25.0452, 121.5168），请计算中国台北车站至中国香港红磡车站的距离。

```
1  # ch4_23.py
2  import math
3
4  r = 6371                          # 地球半径
5  x1, y1 = 22.2838, 114.1731        # 中国香港红磡车站经纬度
6  x2, y2 = 25.0452, 121.5168        # 中国台北车站经纬度
7
8  d = 6371*math.acos(math.sin(math.radians(x1))*math.sin(math.radians(x2))+
9                     math.cos(math.radians(x1))*math.cos(math.radians(x2))*
10                    math.cos(math.radians(y1-y2)))
11
12 print("distance = ", d)
```

执行结果

```
====================== RESTART: D:\Python\ch4\ch4_23.py ======================
distance =  808.3115099471376
```

习题

1. 请重新设计第 2 章的习题 4，将输出方式改为下列方式。(4-2 节)

```
====================== RESTART: D:/Python/ex/ex4_1.py ======================
苹果可以吃 4 天
第 5 天产生苹果不足供应
不足 15 个
```

2. 扩充 ch4_10.py，最右边增加平均分数字段，这个字段的格式化方式是 %4.1f，相当于取到小数第 1 位。(4-2 节)

```
====================== RESTART: D:/Python/ex/ex4_2.py ======================
 姓名    语文    英文    总分    平均
洪冰儒    98      90     188     94.0
洪雨星    96      95     191     95.5
洪冰雨    92      88     180     90.0
洪星宇    93      97     190     95.0
```

3. 设计前一个程序，输出到 out.txt，最后用记事本显示执行结果，下列是执行画面。(4-3 节)

```
====================== RESTART: D:/Python/ex/ex4_3.py ======================
>>>
```

下列是验证 out.txt 结果。

```
  Name     Math    Eng.    Total    Ave.
 Ivan H     98      90      188     94.0
 Univ H     96      95      191     95.5
 Ice Ra     92      88      180     90.0
 Ira Hu     93      97      190     95.0
```

4. 写一个程序，要求用户输入 3 位数数字，最后舍去个位数字输出，例如，输入是 777 输出是 770，输入是 879 输出是 870。(4-4 节)

```
====================== RESTART: D:/Python/ex/ex4_4.py ======================
请输入3位数数字：777
执行结果: 770
>>>
====================== RESTART: D:/Python/ex/ex4_4.py ======================
请输入3位数数字：879
执行结果: 870
```

5. 请重新设计 ch4_20.py，改为输入摄氏温度，转成华氏温度输出，输出温度格式化到小数第 1 位。(4-4 节)

```
====================== RESTART: D:/Python/ex/ex4_5.py ======================
请输入摄氏温度：31
摄氏 31 等于华氏 87.8
```

6. 输入厘米，转成英寸输出，输出格式化到小数第 1 位。提示：1 英寸约是 2.54 厘米。(4-4 节)

```
====================== RESTART: D:\Python\ex\ex4_6.py ======================
请输入厘米：100
厘米 100 等于英寸 39.4
```

7. 输入英寸，转成厘米输出，输出格式化到小数第 1 位。提示：1 英寸约是 2.54 厘米。(4-4 节)

```
====================== RESTART: D:\Python\ex\ex4_7.py ======================
请输入英寸：10
英寸 10 等于厘米 25.4
```

8. 请重新设计 ch2_5.py，将**年利率和存款年数**改为从屏幕输入，输出金额舍去小数相当于单位是元。(4-4 节)

```
====================== RESTART: D:/Python/ex/ex4_8.py ======================
请输入年利率%为单位：1.5
请输入年数        ：5
5 年后本金和是 53864
```

9. 请重新设计第 2 章的习题 5，将**火箭飞行速度**改为从屏幕输入，输出舍去小数。(4-4 节)

```
====================== RESTART: D:/Python/ex/ex4_9.py ======================
请输入火箭速度每分钟千米数：400
地球到月球所需分钟总数 961
```

10. 请重新设计 ch3_24.py，将速度 speed 改为从屏幕输入**马赫数**，程序会将速度马赫数转为千米 / 小时，然后才开始运算。(4-4 节)

```
===================== RESTART: D:/Python/ex/ex4_10.py =====================
请输入火箭速度马赫数：1
总共需要13天, 1小时
>>>
===================== RESTART: D:/Python/ex/ex4_10.py =====================
请输入火箭速度马赫数：3
总共需要4天, 8小时
```

11. 请重新设计程序实例 ch3_26.py，计算两个点之间的距离，但是将点的坐标改为从屏幕输入，一行需可以输入 x 和 y 坐标，输出到小数第 2 位。(4-5 节)

```
===================== RESTART: D:/Python/ex/ex4_11.py =====================
请输入第 1 个点的 x,y 坐标 : 1, 8
请输入第 2 个点的 x,y 坐标 : 3, 10
两点的距离是 : 2.83
```

12. 前一个习题的扩充，平面任意 3 个点可以产生三角形，请输入任意 3 个点的坐标，可以使用下列公式计算此三角形的面积。假设三角形各边长是 dist1、dist2、dist3。(4-5 节)

$$p = (dist1 + dist2 + dist3) / 2$$

$$area = \sqrt{p(p-dist1)(p-dist2)(p-dist3)}$$

```
===================== RESTART: D:/Python/ex/ex4_12.py =====================
请输入第1个点的 x,y 坐标 : 1.5, 5.5
请输入第2个点的 x,y 坐标 : -2.1, 4
请输入第3个点的 x,y 坐标 : -8, -3.2
三角形面积是 : 8.54
```

13. 在 4-7-3 节介绍了正五角形的面积计算公式，可以将该公式扩充为正多边形面积计算，如下所示。(4-7 节)

$$\textbf{area} = \frac{n \times s^2}{4 \times \tan\left(\frac{\pi}{n}\right)}$$

```
====================== RESTART: D:/Python/ex/ex4_13.py ======================
请输入正多边形边数 : 4
请输入正多边形边长 : 4
area =  16.000000000000004
>>>
====================== RESTART: D:/Python/ex/ex4_13.py ======================
请输入正多边形边数 : 5
请输入正多边形边长 : 5
area =  43.01193501472417
>>>
====================== RESTART: D:/Python/ex/ex4_13.py ======================
请输入正多边形边数 : 6
请输入正多边形边长 : 6
area =  93.53074360871938
```

14. 请扩充 ch4_23.py，将程序改为输入两个地点的经纬度，本程序可以计算这两个地点的距离。(4-7 节)

```
====================== RESTART: D:/Python/ex/ex4_14.py ======================
请输入第一个地点的经纬度 : 22.0652, 114.3457
请输入第二个地点的经纬度 : 24.7667, 121.5966
distance =  798.3475207412483
```

15. 假设一架飞机起飞的速度是 v，飞机的加速度是 a，下列是飞机起飞时所需的跑道长度公式。(4-7 节)

$$\text{distance} = \frac{v^2}{2a}$$

请输入飞机时速（米 / 秒）和加速速（米 / 秒），然后列出所需跑道长度（米）。

```
====================== RESTART: D:/Python/ex/ex4_15.py ======================
请输入加速度 a 和速度 v : 3, 80
所需跑道长度 1066.7 米
```

16. 北京故宫博物院的经纬度信息大约是（39.9196, 116.3669），法国巴黎罗浮宫的经纬度大约是（48.8595, 2.3369），请计算这两个博物馆之间的距离。(4-7 节)

```
====================== RESTART: D:/Python/ex/ex4_16.py ======================
distance =  8214.08589098231
```

05

第 5 章

流程控制及 if 语句的使用

本章摘要

一个程序如果是按部就班从头到尾，中间没有转折，其实是无法完成太多工作的。程序设计过程中难免会需要转折，这个转折在程序设计中的术语是**流程控制**。本章将完整讲解有关 if 语句的流程控制。另外，与程序流程设计有关的**关系运算符**与**逻辑运算符**也将在本章做说明，因为这些是 if 语句流程控制的基础。

5-1 关系运算符

Python 语言所使用的关系运算如下。

关系运算符	说明	实例	说明
>	大于	a > b	检查是否 a 大于 b
>=	大于或等于	a >= b	检查是否 a 大于或等于 b
<	小于	a < b	检查是否 a 小于 b
<=	小于或等于	a <= b	检查是否 a 小于或等于 b
==	等于	a == b	检查是否 a 等于 b
!=	不等于	a != b	检查是否 a 不等于 b

上述运算如果是**真**会返回 True，如果是**伪**会返回 False。

实例 1：下列会返回 True。

```
>>> x = 10 > 8
>>> print(x)
True
>>> x = 10 >= 10
>>> print(x)
True
>>> x = 10 < 20
>>> print(x)
True
>>> x = 10 <= 10
>>> print(x)
True
>>> x = 10 == 10
>>> print(x)
True
>>> x = 10 != 20
>>> print(x)
True
>>>
```

实例 2：下列会返回 False。

```
>>> x = 10 > 20
>>> print(x)
False
>>> x = 10 >= 20
>>> print(x)
False
>>> x = 10 < 5
>>> print(x)
False
>>> x = 10 <= 5
>>> print(x)
False
>>> x = 10 == 5
>>> print(x)
False
>>> x = 10 != 10
>>> print(x)
False
>>>
```

5-2 逻辑运算符

Python 所使用的逻辑运算符有以下三个。

and：相当于逻辑符号 AND。

or：相当于逻辑符号 OR。

not：相当于逻辑符号 NOT。

下列是逻辑运算符 and 的图例说明。

and	True	False
True	True	False
False	False	False

实例 1：下列会返回 True。

```
>>> x = (10 > 8) and (20 > 10)
>>> print(x)
True
>>>
```

实例 2：下列会返回 False。

```
>>> x = (10 > 8) and (10 > 20)
>>> print(x)
False
>>> x = (10 < 8) and (10 < 20)
>>> print(x)
False
>>> x = (10 < 8) and (10 > 20)
>>> print(x)
False
>>>
```

下列是逻辑运算符 or 的图例说明。

or	True	False
True	True	True
False	True	False

实例 3：下列会返回 True。

```
>>> x = (10 > 8) or (20 > 10)
>>> print(x)
True
>>> x = (10 < 8) or (10 < 20)
>>> print(x)
True
>>> x = (10 > 8) or (10 > 20)
>>> print(x)
True
>>> .
```

实例 4：下列会返回 False。

```
>>> x = (10 < 8) or (10 > 20)
>>> print(x)
False
>>> .
```

下列是逻辑运算符 not 的图例说明。

not	True	False
	False	True

如果是 True 经过 not 运算会返回 False，如果是 False 经过 not 运算会返回 True。

实例 5：下列会返回 True。

```
>>> x = not(10 < 8)
>>> print(x)
True
>>>
```

实例 6：下列会返回 False。

```
>>> x = not(10 > 8)
>>> print(x)
False
>>>
```

5-3 if 语句

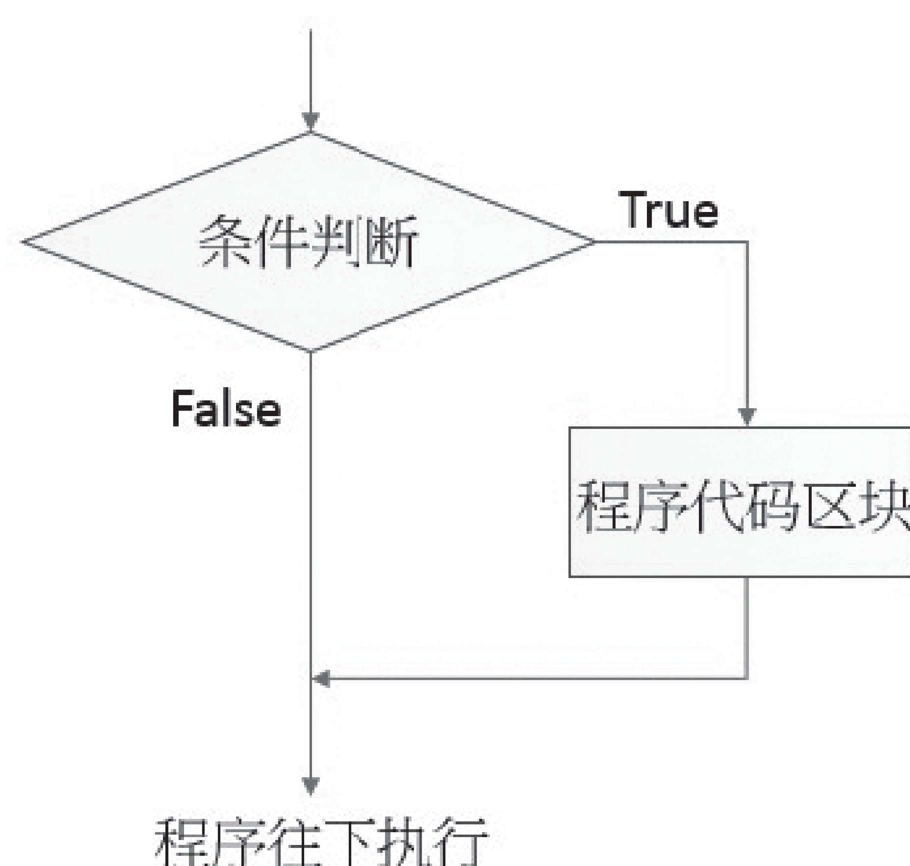
if 语句的基本语法如下：

```
if （条件判断）:                # 条件判断外的小括号可有可无
程序代码区块
```

如果**条件判断**是 True，则**执行程序代码区块**，如果**条件判断**是 False，则**不执行程序代码区块**。如果程序代码区块只有一条指令，可将上述语法写成下列格式。

```
if （条件判断）: 程序代码区块
```

可以用下列流程图说明这个 if 语句。



如果读者学习过其他程序语言，例如 Visual Basic、C、JavaScript 等，在条件表达式中是使用大括号“{ }”，将 if 语句的程序代码区块括起来，如下所示（以 C 语言为实例）。

```
if (age < 20) {
    printf("你年龄太小");
    printf("须年满 20 岁才可购买烟酒");
}
```

在 Python 内是使用**内缩方式**区隔 if 语句的程序代码区块，编辑程序时可以用 Tab 键内缩或是直接**内缩 4 个字符空间**，表示这是 if 语句的程序代码区块。相同内容，可以用下列方式处理。

```
If (age < 20):                  # 程序代码区块 1
   print("你年龄太小")           # 程序代码区块 2
```

print(" 须年满 20 岁才可购买烟酒 ")　　　　　# 程序代码区块 2

在 Python 中内缩程序代码是有意义的，相同的程序代码区块，必须有相同的内缩，否则会产生错误。

实例 1：正确的 if 语句程序代码。

```
>>> age = 18
>>> if (age < 20):
        print("你年龄太小")
        print("须年满20岁才可以购买烟酒")
```

插入点在此时请按Enter键

```
>>> age = 18
>>> if (age < 20):
        print("你年龄太小")
        print("须年满20岁才可以购买烟酒")

你年龄太小
须年满20岁才可以购买烟酒
>>>
```

实例 2：不正确的 if 语句程序代码，下列代码因为任意内缩造成错误。

任意内缩造成错误

```
>>> age = 18
>>> if (age < 20):
        print("你年龄太小")
         print("须年满20岁才可以购买烟酒")

SyntaxError: unexpected indent
>>>
```

上述笔者讲解 if 语句是 True 时需内缩 4 个字符空间，这是 Python 预设的，读者可能会问可不可以内缩 5 个字符空间，答案是可以的，但是记得相同程序区块必须有相同的内缩空间。不过如果是使用 Python 的 IDLE 编辑环境，当输入 if 语句后，只要按 Enter 键，程序就会自动内缩 4 个字符空间。

程序实例 ch5_1.py：if 语句的基本应用。

```
1  # ch5_1.py
2  age = input("请输入年龄: ")
3  if (int(age) < 20):
4      print("你年龄太小")
5      print("须年满20岁才可以购买烟酒")
```

执行结果

```
===================== RESTART: D:\Python\ch5\ch5_1.py =====================
请输入年龄: 18
你年龄太小
须年满20岁才可以购买烟酒
>>>
===================== RESTART: D:\Python\ch5\ch5_1.py =====================
请输入年龄: 21
```

程序实例 ch5_2.py：输出绝对值的应用。

```
1  # ch5_2.py
2  print("输出绝对值")
3  num = input("请输入任意整数值: ")
4  x = int(num)
5  if (int(x) < 0):
6      x = abs(x)
7  print("绝对值是 %d" % int(x))
```

执行结果

```
===================== RESTART: D:\Python\ch5\ch5_2.py =====================
输出绝对值
请输入任意整数值: 98
绝对值是 98
>>>
===================== RESTART: D:\Python\ch5\ch5_2.py =====================
输出绝对值
请输入任意整数值: -30
绝对值是 30
```

对于上述 ch5_2.py 而言，由于 if 语句只有一条指令，所以可以将第 5 行和第 6 行改写成下列语句。

```
5  if (int(x) < 0): x = abs(x)
```

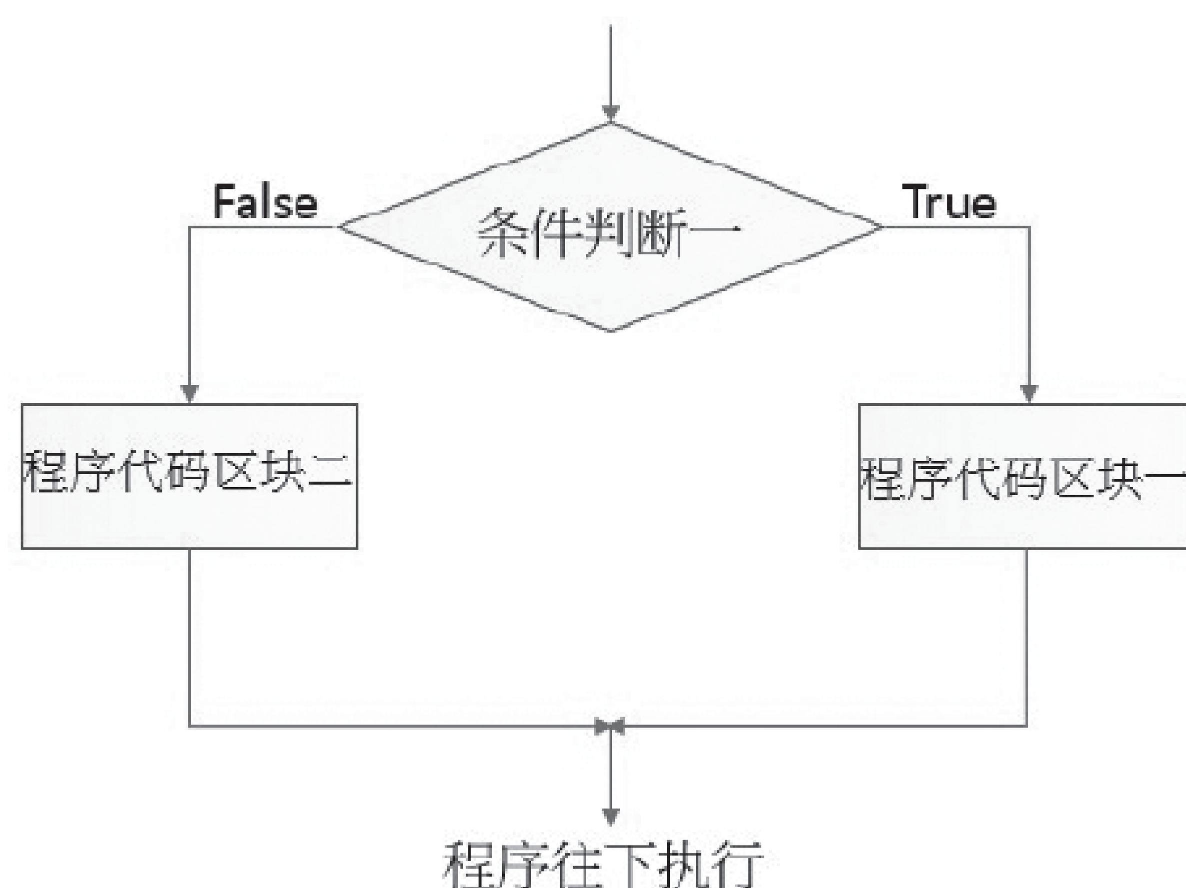
上述可以得到相同的结果，详请可参考本书代码文件中的 ch5_2_1.py。

5-4 if … else 语句

程序设计时更常用的功能是条件判断为 True 时执行某一个程序代码区块，当条件判断为 False 时执行另一段程序代码区块，此时可以使用 if … else 语句，它的语法格式如下：

```
if  （条件判断）:
程序代码区块一
else:
程序代码区块二
```

如果条件判断是 True，则执行程序代码区块一，如果条件判断是 False，则执行程序代码区块二。可以用下列流程图说明这个 if … else 语句。



程序实例 ch5_3.py：重新设计 ch5_1.py，增加年龄满 20 岁时的输出。

```
1  # ch5_3.py
2  age = input("请输入年龄: ")
3  if (int(age) < 20):
4      print("你年龄太小")
5      print("须年满20岁才可以购买烟酒")
6  else:
7      print("欢迎购买烟酒")
```

执行结果

```
===================== RESTART: D:\Python\ch5\ch5_3.py =====================
请输入年龄: 18
你年龄太小
须年满20岁才可以购买烟酒
>>>
===================== RESTART: D:\Python\ch5\ch5_3.py =====================
请输入年龄: 30
欢迎购买烟酒
```

程序实例 ch5_4.py：奇数偶数的判断。

```
1  # ch5_4.py
2  print("奇数偶数判断")
3  num = input("请输入任意整值: ")
4  rem = int(num) % 2
5  if (rem == 0):
6      print("%d 是偶数" % int(num))
7  else:
8      print("%d 是奇数" % int(num))
```

执行结果

```
===================== RESTART: D:\Python\ch5\ch5_4.py =====================
奇数偶数判断
请输入任意整值: 5
5 是奇数
>>>
===================== RESTART: D:\Python\ch5\ch5_4.py =====================
奇数偶数判断
请输入任意整值: 10
10 是偶数
```

Python 语言在执行网络爬虫存取数据时，常会不知道可以获得多少笔数据，例如可能是 0 ～ 100 笔，如果我们想要最多只取 10 笔数据（小于 10 笔也可以当作我们的数据），使用传统程序语言的语法，设计观念应该如下：

```
if items >= 10:
    items = 10
else:
    items = items
```

在 Python 中，我们可以用下列语法表达：

```
items = 10 if items >= 10 else items
```

程序实例 ch5_4_1.py：测试 if … else 语法。

```
1  # ch5_4_1.py
2  items = 5
3  items = 10 if items >= 10 else items
4  print(items)
5  items = 15
6  items = 10 if items >= 10 else items
7  print(items)
```

执行结果

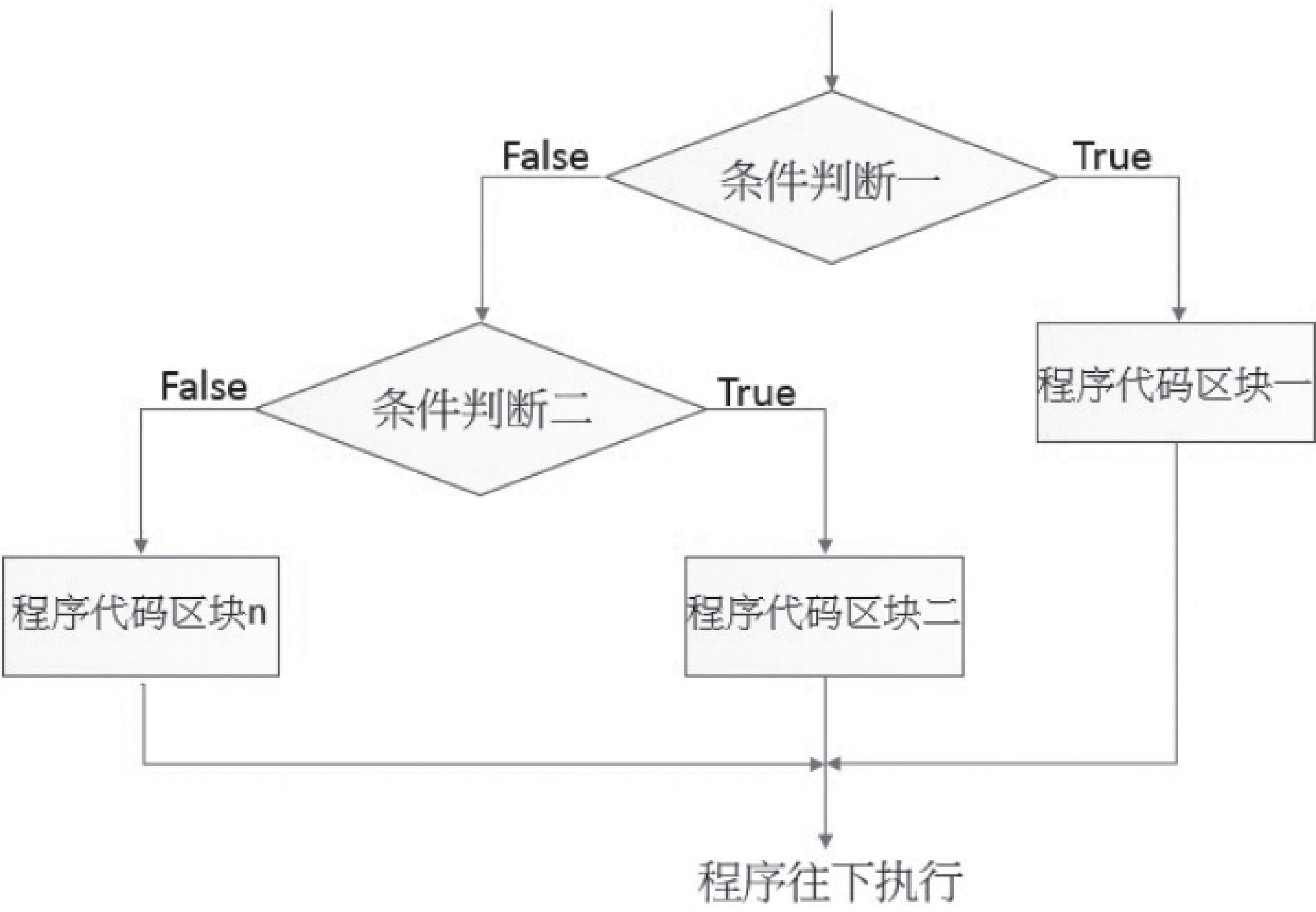
```
===================== RESTART: D:/Python/ch5/ch5_4_1.py =====================
5
10
```

5-5 if … elif … else 语句

这是一个多重判断，程序设计时需要多个条件做比较时就比较有用。例如，在美国成绩计分是采取 A、B、C、D、F 等，通常 90 ～ 100 分是 A，80 ～ 89 分是 B，70 ～ 79 分是 C，60 ～ 69 分是 D，低于 60 分是 F。使用 Python 可以用这个语句，很容易就可以完成这个工作。这个语句的基本语法如下。

```
if  （条件判断一）：
程序代码区块一
elif （条件判断二）：
程序代码区块二
…
else:
程序代码区块 n
```

如果**条件判断一**是 True 则执行**程序代码区块一**，然后离开条件判断。否则检查**条件判断二**，如果是 True 则执行**程序代码区块二**，然后离开条件判断。如果**条件判断**是 False 则持续进行检查，上述 elif 的条件判断可以不断扩充，如果所有条件判断是 False 则执行**程序代码 n 区块**。下列流程图是假设只有两个条件判断说明这个 if … elif … else 语句。



程序实例 ch5_5.py：请输入数字分数，程序将响应 A、B、C、D 或 F 等级。

```
1  # ch5_5.py
2  print("计算最终成绩")
3  score = input("请输入分数: ")
4  sc = int(score)
5  if (sc >= 90):
6      print(" A")
7  elif (sc >= 80):
8      print(" B")
9  elif (sc >= 70):
10     print(" C")
11 elif (sc >= 60):
12     print(" D")
13 else:
14     print(" F")
```

执行结果

```
===================== RESTART: D:\Python\ch5\ch5_5.py =====================
计算最终成绩
请输入分数: 90
 A
>>>
===================== RESTART: D:\Python\ch5\ch5_5.py =====================
计算最终成绩
请输入分数: 83
 B
>>>
===================== RESTART: D:\Python\ch5\ch5_5.py =====================
计算最终成绩
请输入分数: 79
 C
>>>
===================== RESTART: D:\Python\ch5\ch5_5.py =====================
计算最终成绩
请输入分数: 66
 D
>>>
===================== RESTART: D:\Python\ch5\ch5_5.py =====================
计算最终成绩
请输入分数: 55
 F
```

程序实例 ch5_6.py：有一地区的票价收费标准是 100 元。

（1）如果小于等于 6 岁或大于等于 80 岁，收费是打 2 折。

（2）如果是 7 ～ 12 岁或 60 ～ 79 岁，收费是打 5 折。

请输入年龄，程序会计算票价。

```
1  # ch5_6.py
2  print("计算票价")
3  age = input("请输入年龄: ")
4  age = int(age)
5  ticket = 100
6  if age >= 80 or age <= 6:
7      ticket = ticket * 0.2
8      print("票价是: %d" % ticket)
9  elif age >= 60 or age <= 12:
10     ticket = ticket * 0.5
11     print("票价是: %d" % ticket)
12 else:
13     print("票价是: %d" % ticket)
```

执行结果

```
===================== RESTART: D:\Python\ch5\ch5_6.py =====================
计算票价
请输入年龄: 81
票价是: 20
>>>
===================== RESTART: D:\Python\ch5\ch5_6.py =====================
计算票价
请输入年龄: 6
票价是: 20
>>>
===================== RESTART: D:\Python\ch5\ch5_6.py =====================
计算票价
请输入年龄: 77
票价是: 50
>>>
===================== RESTART: D:\Python\ch5\ch5_6.py =====================
计算票价
请输入年龄: 12
票价是: 50
>>>
===================== RESTART: D:\Python\ch5\ch5_6.py =====================
计算票价
请输入年龄: 13
票价是: 100
```

上述程序的第 6 行和第 9 行，如果读者对于运算符执行的优先级没有太大的把握，建议直接用小括号将条件判断括起来，可参考 ch5_6_1.py。

```
6  if (age >= 80) or (age <= 6):
7      ticket = ticket * 0.2
8      print("票价是: %d" % ticket)
9  elif (age >= 60) or (age <= 12):
```

程序实例 ch5_7.py：这个程序要求输入字符，然后告知所输入的字符是大写字母、小写字母、阿拉伯数字或特殊字符。

```
1  # ch5_7.py
2  print("判断输入字符类别")
3  ch = input("请输入字符: ")
4  if ord(ch) >= ord("A") and ord(ch) <= ord("Z"):
5      print("这是大写字符")
6  elif ord(ch) >= ord("a") and ord(ch) <= ord("z"):
7      print("这是小写字符")
8  elif ord(ch) >= ord("0") and ord(ch) <= ord("9"):
9      print("这是数字")
10 else:
11     print("这是特殊字符")
```

执行结果

```
===================== RESTART: D:\Python\ch5\ch5_7.py =====================
判断输入字符类别
请输入字符: K
这是大写字符
>>>
===================== RESTART: D:\Python\ch5\ch5_7.py =====================
判断输入字符类别
请输入字符: m
这是小写字符
>>>
===================== RESTART: D:\Python\ch5\ch5_7.py =====================
判断输入字符类别
请输入字符: 9
这是数字
```

5-6 嵌套的 if 语句

嵌套的 if 语句是指在 if 语句内又有其他的 if 语句，下列是一种情况的实例。

```
if (条件判断一):
    if (条件判断A):
        程序代码区块A
    else:
        程序代码区块B
else:
    程序代码区块二
```

这应是原先程序代码区块一，结果出现另一个 if 条件判断

其实 Python 允许加上许多层，不过层次太多时，未来程序维护会变得比较困难。

程序实例 ch5_8.py：测试某一年是否闰年，闰年的条件是首先可以被 4 整除（相当于没有余数），这个条件成立时，还必须符合除以 100 时余数不为 0 或是除以 400 时余数为 0，当两个条件都符合才算闰年。

```
1  # ch5_8.py
2  print("判断输入年份是否闰年")
3  year = input("请输入年份: ")
4  rem4 = int(year) % 4
5  rem100 = int(year) % 100
6  rem400 = int(year) % 400
7  if rem4 == 0:
8      if rem100 != 0 or rem400 == 0:
9          print("%s 是闰年" % year)
10     else:
11         print("%s 不是闰年" % year)
12 else:
13     print("%s 不是闰年" % year)
```

执行结果

```
===================== RESTART: D:\Python\ch5\ch5_8.py =====================
判断输入年份是否闰年
请输入年份: 2018
2018 不是闰年
>>>
===================== RESTART: D:\Python\ch5\ch5_8.py =====================
判断输入年份是否闰年
请输入年份: 2020
2020 是闰年
>>>
===================== RESTART: D:\Python\ch5\ch5_8.py =====================
判断输入年份是否闰年
请输入年份: 2100
2100 不是闰年
```

5-7 尚未设置的变量值 None

有人在设计程序时，喜欢将所有变量一次先予以定义，在尚未用到此变量时先设置这个变量的值是 None，如果此时用 type( ) 函数了解它的类型时将显示 NoneType，如下所示。

```
>>> x = None
>>> print(x)
None
>>> type(x)
<class 'NoneType'>
>>>
```

通常在设计程序时，可使用下列方式测试。

程序设计 ch5_8_1.py：if 语句与 None 的应用。不过要注意的是，None 在布尔值运算时会被当作 False。

```
1  # ch5_8_1.py
2  flag = None
3  if flag == None:
4      print("尚未定义flag")
5
6  if flag:
7      print("有定义")
8  else:
9      print("尚未定义flag")
```

执行结果

```
==================== RESTART: D:\Python\ch5\ch5_8_1.py ====================
尚未定义flag
尚未定义flag
```

5-8 专题——BMI 程序 / 猜出生日期 / 十二生肖系统 / 线性方程式

5-8-1　设计人体体重健康判断程序

BMI（Body Mass Index）又称**身高体重指数（也称身体质量指数）**，是由比利时的科学家**凯特勒**（Lambert Quetelet）最先提出，也是世界卫生组织认可的健康指数，它的计算方式如下：

$$BMI = 体重（kg）/ 身高（m）^2$$

如果 BMI 为 18.5 ～ 23.9，表示这是健康的 BMI 值。请输入自己的身高和体重，然后列出是否在健康的范围。中国官方针对 BMI 指数公布的更进一步资料如下。

分类	BMI
体重过轻	BMI < 18.5
正常	18.5 ⩽ BMI and BMI < 24
超重	24 ⩽ BMI and BMI < 28
肥胖	BMI ⩾ 28

程序实例 ch5_9.py：人体健康体重指数判断程序，这个程序会要求输入身高与体重，然后计算 BMI 指数，由这个 BMI 指数判断体重是否正常。

```
1  # ch5_9.py
2  height = input("请输入身高(厘米)：")
3  weight = input("请输入体重(千克)：")
4  bmi = int(weight) / ( (float(height) / 100) ** 2 )
5  if bmi >= 18.5 and bmi < 24:
6      print("体重正常")
7
8  else:
9      print("体重不正常")
```

执行结果

```
===================== RESTART: D:\Python\ch5\ch5_9.py =====================
请输入身高(厘米)：170
请输入体重(千克)：60
体重正常
>>>
===================== RESTART: D:\Python\ch5\ch5_9.py =====================
请输入身高(厘米)：170
请输入体重(千克)：100
体重不正常
>>>
===================== RESTART: D:\Python\ch5\ch5_9.py =====================
请输入身高(厘米)：170
请输入体重(千克)：47
体重不正常
```

上述程序第 4 行 "float（height）/100"，主要是将身高单位由厘米改为米，上述专题程序可以扩充为输入身高体重后，程序可以列出相应 BMI 值及其所在区间，作为读者的习题。

5-8-2　猜出生日期

本节将先说明程序，随后再说明程序的工作原理。在讲解猜出生日期之前，先用更简单的猜 0 ～ 7 数字做说明。

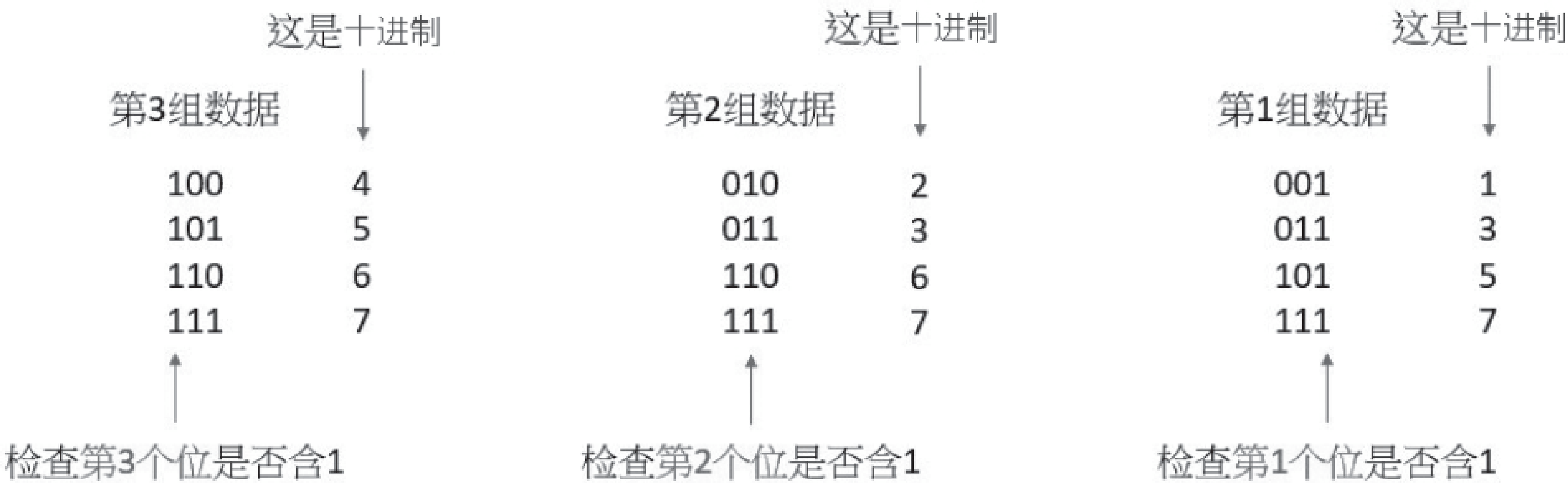
程序实例 ch5_10.py：读者心中先预想一个 0 ～ 7 的数字，程序中会问读者 3 个问题，请读者真心回答，然后这个程序会猜出读者心中的数字。

```
1   # ch5_10.py
2   ans = 0                                  # 读者心中的数字
3   print("猜数字游戏,请心中想一个 0 ~ 7之间的数字, 然后回答问题")
4
5   truefalse = "输入y或Y代表有, 其他代表无 : "
6   # 检测二进制的第1位是否含1
7   q1 = "有没有看到心中的数字 : \n" + \
8         "1, 3, 5, 7 \n"
9   num = input(q1 + truefalse)
10  print(num)
11  if num == "y" or num == "Y":
12      ans += 1
13  # 检测二进制的第2位是否含1
14  truefalse = "输入y或Y代表有, 其他代表无 : "
15  q2 = "有没有看到心中的数字 : \n" + \
16        "2, 3, 6, 7 \n"
17  num = input(q2 + truefalse)
18  if num == "y" or num == "Y":
19      ans += 2
20  # 检测二进制的第3位是否含1
21  truefalse = "输入y或Y代表有, 其他代表无 : "
22  q3 = "有没有看到心中的数字 : \n" + \
23        "4, 5, 6, 7 \n"
24  num = input(q3 + truefalse)
25  if num == "y" or num == "Y":
26      ans += 4
27
28  print("读者心中所想的数字是 : ", ans)
```

执行结果

```
===================== RESTART: D:\Python\ch5\ch5_10.py =====================
猜数字游戏,请心中想一个 0 - 7之间的数字, 然后回答问题
有没有看到心中的数字 :
1, 3, 5, 7
输入y或Y代表有, 其他代表无 : n
n
有没有看到心中的数字 :
2, 3, 6, 7
输入y或Y代表有, 其他代表无 : y
有没有看到心中的数字 :
4, 5, 6, 7
输入y或Y代表有, 其他代表无 : y
读者心中所想的数字是 :  6
```

0 ～ 7 的数字基本上可用 3 个二进制表示，为 000 ～ 111。其实所问的 3 个问题，基本上只是了解特定位是否为 1。



了解了以上概念，我们可以再进一步扩充上述实例猜测一个人生日的日期，一个人生日的日期是 1 ～ 31 的数字。

程序实例 ch5_11.py：猜测一个人生日的日期，对于 1 ～ 31 的数字可以用 5 个二进制的位表示，所以可以询问 5 个问题，每个问题获得一个位是否为 1，经过 5 个问题即可获得一个人的生日日期，下列是 5 组数据信息。

第5组数据	这是十进制	第4组数据	这是十进制	第3组数据	这是十进制	第2组数据	这是十进制	第1组数据	这是十进制
10000	16	01000	8	00100	4	00010	2	00001	1
10001	17	01001	9	00101	5	00011	3	00011	3
10010	18	01010	10	00110	6	00110	6	00101	5
10011	19	01011	11	00111	7	00111	7	00111	7
10100	20	01100	12	01100	12	01010	10	01001	9
10101	21	01101	13	01101	13	01011	11	01011	11
10110	22	01110	14	01110	14	01110	14	01101	13
10111	23	01111	15	01111	15	01111	15	01111	15
11000	24	11000	24	10100	20	10010	18	10001	17
11001	25	11001	25	10101	21	10011	19	10011	19
11010	26	11010	26	10110	22	10110	22	10101	21
11011	27	11011	27	10111	23	10111	23	10111	23
11100	28	11100	28	11100	28	11010	26	11001	25
11101	29	11101	29	11101	29	11011	27	11011	27
11110	30	11110	30	11110	30	11110	30	11101	29
11111	31	11111	31	11111	31	11111	31	11111	31

```
1  # ch5_11.py
2  ans = 0                          # 读者心中的数字
3  print("猜生日日期游戏,请回答下列5个问题,这个程序即可列出你的生日")
4
5  truefalse = "输入y或Y代表有, 其他代表无 : "
6  # 检测二进制的第1位是否含1
7  q1 = "有没有看到自己的生日日期 : \n" + \
8        "1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31 \n"
9  num = input(q1 + truefalse)
10 print(num)
11 if num == "y" or num == "Y":
12     ans += 1
13 # 检测二进制的第2位是否含1
14 truefalse = "输入y或Y代表有, 其他代表无 : "
15 q2 = "有没有看到自己的生日日期 : \n" + \
16       "2, 3, 6, 7, 10, 11, 14, 15, 18, 19, 22, 23, 26, 27, 30, 31 \n"
17 num = input(q2 + truefalse)
18 if num == "y" or num == "Y":
19     ans += 2
20 # 检测二进制的第3位是否含1
21 truefalse = "输入y或Y代表有, 其他代表无 : "
22 q3 = "有没有看到自己的生日日期 : \n" + \
23       "4, 5, 6, 7, 12, 13, 14, 15, 20, 21, 22, 23, 28, 29, 30, 31 \n"
24 num = input(q3 + truefalse)
25 if num == "y" or num == "Y":
26     ans += 4
27 # 检测二进制的第4位是否含1
28 truefalse = "输入y或Y代表有, 其他代表无 : "
29 q4 = "有没有看到自己的生日日期 : \n" + \
30       "8, 9, 10, 11, 12, 13, 14, 15, 24, 25, 26, 27, 28, 29, 30, 31 \n"
31 num = input(q4 + truefalse)
32 if num == "y" or num == "Y":
33     ans += 8
34 # 检测二进制的第5位是否含1
35 truefalse = "输入y或Y代表有, 其他代表无 : "
36 q5 = "有没有看到自己的生日日期 : \n" + \
37       "16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31 \n"
38 num = input(q5 + truefalse)
39 if num == "y" or num == "Y":
40     ans += 16
41
42 print("读者的生日日期是 : ", ans)
```

执行结果

```
======================= RESTART: D:\Python\ch5\ch5_11.py =======================
猜生日日期游戏,请回答下列5个问题,这个程序即可列出你的生日
有没有看到自己的生日日期 :
1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31
输入y或Y代表有, 其他代表无 : n
n
有没有看到自己的生日日期 :
2, 3, 6, 7, 10, 11, 14, 15, 18, 19, 22, 23, 26, 27, 30, 31
输入y或Y代表有, 其他代表无 : n
有没有看到自己的生日日期 :
4, 5, 6, 7, 12, 13, 14, 15, 20, 21, 22, 23, 28, 29, 30, 31
输入y或Y代表有, 其他代表无 : y
有没有看到自己的生日日期 :
8, 9, 10, 11, 12, 13, 14, 15, 24, 25, 26, 27, 28, 29, 30, 31
输入y或Y代表有, 其他代表无 : y
有没有看到自己的生日日期 :
16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31
输入y或Y代表有, 其他代表无 : n
读者的生日日期是 :  12
```

5-8-3　十二生肖系统

在中国除了使用公元年份代号，也使用鼠、牛、虎、兔、龙、蛇、马、羊、猴、鸡、狗、猪当

作十二生肖，每 12 年是一个周期，1900 年是鼠年。

程序实例 ch5_12.py：请输入你出生的公元年 19×× 或 20××，本程序会输出相对应的生肖年。

```
1  # ch5_12.py
2  year = eval(input("请输入公元出生年 : "))
3  year -= 1900
4  zodiac = year % 12
5  if zodiac == 0:
6      print("你的生肖是 : 鼠")
7  elif zodiac == 1:
8      print("你的生肖是 : 牛")
9  elif zodiac == 2:
10     print("你的生肖是 : 虎")
11 elif zodiac == 3:
12     print("你的生肖是 : 兔")
13 elif zodiac == 4:
14     print("你的生肖是 : 龙")
15 elif zodiac == 5:
16     print("你的生肖是 : 蛇")
17 elif zodiac == 6:
18     print("你的生肖是 : 马")
19 elif zodiac == 7:
20     print("你的生肖是 : 羊")
21 elif zodiac == 8:
22     print("你的生肖是 : 猴")
23 elif zodiac == 9:
24     print("你的生肖是 : 鸡")
25 elif zodiac == 10:
26     print("你的生肖是 : 狗")
27 else:
28     print("你的生肖是 : 猪")
```

执行结果

```
===================== RESTART: D:\Python\ch5\ch5_12.py =====================
请输入公元出生年 : 1961
你的生肖是 : 牛
>>>
===================== RESTART: D:\Python\ch5\ch5_12.py =====================
请输入公元出生年 : 1975
你的生肖是 : 兔
```

注　以上是用公元日历，十二生肖年是用农历年，所以年初或年尾会有一些差异。

5-8-4　求一元二次方程式的根

在中学数学中，可以看到下列一元二次方程式：

$$ax^2 + bx + c = 0$$

可以用下列方式获得根。

$$\mathbf{r1} = \frac{-b+\sqrt{b^2-4ac}}{2a} \qquad \mathbf{r2} = \frac{-b-\sqrt{b^2-4ac}}{2a}$$

上述方程式有 3 种状况，如果 $b^2 - 4ac$ 是正值，那么这个一元二次方程式有两个实数根。如果 $b^2 - 4ac$ 是 0，那么这个一元二次方程式有一个实数根。如果 $b^2 - 4ac$ 是负值，那么这个一元二次方程式没有实数根。

实数根的几何意义是与 x 轴交叉点的坐标。

程序实例 ch5_13.py：有一个一元二次方程式如下：

$$3x^2 + 5x + 1 = 0$$

求这个方程式的根。

```
1  # ch5_13.py
2  a = 3
3  b = 5
4  c = 1
5
6  r1 = (-b + (b**2-4*a*c)**0.5)/(2*a)
7  r2 = (-b - (b**2-4*a*c)**0.5)/(2*a)
8  print("r1 = %6.4f,     r2 = %6.4f" % (r1, r2))
```

执行结果

```
====================== RESTART: D:\Python\ch5\ch5_13.py ======================
r1 = -0.2324,     r2 = -1.4343
```

5-8-5 求解联立线性方程式

假设有一个联立线性方程式如下：

$$ax + by = e$$

$$cx + dy = f$$

可以用下列方式获得 x 和 y 值。

$$x = \frac{e\times d - b\times f}{a\times d - b\times c} \quad y = \frac{a\times f - e\times c}{a\times d - b\times c}$$

在上述公式中，如果 $a\times d - b\times c$ 等于 0，则此联立线性方程式无解。

程序实例 ch5_14.py：计算下列联立线性方程式的值。

$$2x + 3y = 13$$

$$x - 2y = -4$$

```
1  # ch5_14.py
2  a = 2
3  b = 3
4  c = 1
5  d = -2
6  e = 13
7  f = -4
8
9  x = (e*d - b*f) / (a*d - b*c)
10 y = (a*f - e*c) / (a*d - b*c)
11 print("x = %6.4f,     y = %6.4f" % (x, y))
```

执行结果

```
====================== RESTART: D:/Python/ch5/ch5_14.py ======================
x = 2.0000,     y = 3.0000
```

习题

1. 请改为不使用 abs( ) 函数重新设计 ch5_2.py 程序。(5-3 节)

```
====================== RESTART: D:/Python/ex/ex5_1.py ======================
输出绝对值
请输入任意整数值: -99
绝对值是 99
>>>
====================== RESTART: D:/Python/ex/ex5_1.py ======================
输出绝对值
请输入任意整数值: 88
绝对值是 88
```

2. 请输入 3 个数字，本程序可以将数字由大到小输出。(5-3 节)

```
====================== RESTART: D:/Python/ex/ex5_2.py ======================
请输入3个整数值: 3, 6, 5
大到小分别是  6 5 3
>>>
====================== RESTART: D:/Python/ex/ex5_2.py ======================
请输入3个整数值: 2, 8, 10
大到小分别是  10 8 2
```

3. 有一个圆半径是 20，圆中心在坐标（0,0）位置，请输入任意点坐标，这个程序可以判断此点坐标是不是在圆内部。(5-4 节)

提示　可以计算点坐标距离圆中心的长度是否小于半径。

```
====================== RESTART: D:/Python/ex/ex5_3.py ======================
请输入点坐标 : 10, 10
点坐标(10,10)在圆内部
>>>
====================== RESTART: D:/Python/ex/ex5_3.py ======================
请输入点坐标 : 20, 20
点坐标(20,20)不在圆内部
```

4. 请设计一个程序，如果输入是负值则将它改成正值输出，如果输入是正值则将它改成负值输出。(5-4 节)

```
====================== RESTART: D:/Python/ex/ex5_4.py ======================
输入数字判断程序
请输入任意整数值: 5
-5
>>>
====================== RESTART: D:/Python/ex/ex5_4.py ======================
输入数字判断程序
请输入任意整数值: -9
9
>>>
====================== RESTART: D:/Python/ex/ex5_4.py ======================
输入数字判断程序
请输入任意整数值: 0
0
```

5. 用户可以先选择华氏温度与摄氏温度转换方式，然后输入一个温度，可以转换成另一种温度。(5-5 节)

```
====================== RESTART: D:/Python/ex/ex5_5.py ======================
温度转换选择
1:华氏温度转成摄氏温度
2:摄氏温度转华氏温度
= 1
请输入华氏温度：104
华氏 104 等于摄氏 40.0
>>>
====================== RESTART: D:/Python/ex/ex5_5.py ======================
温度转换选择
1:华氏温度转成摄氏温度
2:摄氏温度转华氏温度
= 2
请输入摄氏温度：31
摄氏 31 等于华氏 87.8
>>>
====================== RESTART: D:/Python/ex/ex5_5.py ======================
温度转换选择
1:华氏温度转成摄氏温度
2:摄氏温度转华氏温度
= 3
输入错误
```

6. 假设在麦当劳打工每周领一次薪资，工作基本时薪是 150 元，其他规则如下。

（1）小于 40 小时（周），每小时是基本时薪的 0.8 倍。

（2）等于 40 小时（周），每小时是基本时薪。

（3）大于 40 至 50（含）小时（周），每小时是基本时薪的 1.2 倍。

（4）大于 50 小时（周），每小时是基本时薪的 1.6 倍。

请输入工作时数，然后可以计算周薪。(5-5 节)

```
====================== RESTART: D:/Python/ex/ex5_6.py ======================
请输入本周工作时数 : 20
本周薪资 : 2400
>>>
====================== RESTART: D:/Python/ex/ex5_6.py ======================
请输入本周工作时数 : 40
本周薪资 : 6000
>>>
====================== RESTART: D:/Python/ex/ex5_6.py ======================
请输入本周工作时数 : 45
本周薪资 : 8100
>>>
====================== RESTART: D:/Python/ex/ex5_6.py ======================
请输入本周工作时数 : 60
本周薪资 : 14400
```

7. 假设今天是星期日，请输入天数 days，本程序可以响应 days 天后是星期几。(5-5 节)

```
====================== RESTART: D:/Python/ex/ex5_7.py ======================
今天是星期日
请输入天数 : 5
5 天后是星期五
>>>
====================== RESTART: D:/Python/ex/ex5_7.py ======================
今天是星期日
请输入天数 : 10
10 天后是星期三
```

8. 三角形边长的要求是两边长加起来大于第三边，请输入 3 个边长，如果这 3 个边长可以形成三角形则输出三角形的周长。如果这 3 个边长无法形成三角形，则输出这不是三角形的边长。(5-6 节)

```
====================== RESTART: D:/Python/ex/ex5_8.py ======================
请输入3边长 : 3, 3, 3
三角形周长是 :  9
>>>
====================== RESTART: D:/Python/ex/ex5_8.py ======================
请输入3边长 : 3, 3, 9
这不是三角形的边长
```

9. 扩充设计 ch5_9.py，列出中国 BMI 指数区分的结果表。(5-7 节)

```
====================== RESTART: D:/Python/ex/ex5_9.py ======================
请输入身高(厘米):170
请输入体重(千克):49
体重过轻
>>>
====================== RESTART: D:/Python/ex/ex5_9.py ======================
请输入身高(厘米):170
请输入体重(千克):62
正常
>>>
====================== RESTART: D:/Python/ex/ex5_9.py ======================
请输入身高(厘米):170
请输入体重(千克):80
超重
>>>
====================== RESTART: D:/Python/ex/ex5_9.py ======================
请输入身高(厘米):170
请输入体重(千克):90
肥胖
```

10. 请参考 ch5_13.py，但是修改为在屏幕上输入 a, b, c 三个数值，彼此用逗号隔开，然后计算此一元二次方程式的根，先列出有几个根。如果有实数根则列出根值，如果没有实数根则列出没有

实数根，然后程序结束。(5-7 节)

```
===================== RESTART: D:/Python/ex/ex5_10.py =====================
请输入一元二次方程式的系数 : 3, 5, 1
有2个根
r1 = -0.2324,    r2 = -1.4343
>>>
===================== RESTART: D:/Python/ex/ex5_10.py =====================
请输入一元二次方程式的系数 : 1, 2, 1
有1个根
r1 = -1.0000
>>>
===================== RESTART: D:/Python/ex/ex5_10.py =====================
请输入一元二次方程式的系数 : 1, 2, 8
没有实数根
```

11. 请参考 ch5_14.py，但是修改为在屏幕上输入 a, b, c, d, e, f 六个数值，彼此用逗号隔开，这些数值分别是联立线性方程式的系数与方程式的值，然后计算此线性方程式的 x 和 y 值，如果此题无解则列出此题目没有解。(5-7 节)

```
===================== RESTART: D:/Python/ex/ex5_11.py =====================
请输入线性方程式的系数 : 2, 3, 1, -2, 13, -4
x = 2.0000,    y = 3.0000
>>>
===================== RESTART: D:/Python/ex/ex5_11.py =====================
请输入线性方程式的系数 : 1, 2, 2, 4, 4, 5
此线性方程式没有解
```

06

第 6 章

列表

本章摘要

6-1　认识列表
6-2　Python 简单的面向对象概念
6-3　获得列表的方法
6-4　增加与删除列表元素
6-5　列表的排序
6-6　进阶列表操作
6-7　列表内含列表
6-8　列表的赋值与切片复制
6-9　再谈字符串
6-10　in 和 not in 表达式
6-11　is 和 is not 表达式
6-12　enumerate 对象
6-13　专题——建立大型列表 / 用户账号管理系统 / 文件加密

列表（list）是 Python 中一种可以更改内容的数据类型，它是由一系列元素所组成的序列。如果现在要设计班上同学的成绩表，班上有 50 位同学，可能需要设计 50 个变量，这是一件麻烦的事。如果学校要设计所有学生的数据库，学生人数有 1000 人，需要 1000 个变量，这似乎是不可能的事。Python 的**列表**数据类型，可以只用一个变量，解决这方面的问题，存取时用**列表名称**加上**索引值**即可，这也是本章的主题。

相信阅读至此章节，读者已经对 Python 有一些基础知识了，本章将讲解简单的**面向对象**的概念，同时教导读者学习利用 Python 所提供的内建资源，一步一步带领读者迈向高手之路。

6-1　认识列表

其实在其他程序语言中，与列表相类似的功能称为**数组**(array)，例如 C 语言。不过，Python 的**列表**功能除了可以存储相同数据类型，例如**整数**、**浮点数**、**字符串**（我们将每一笔数据称为**元素**），也可以存储不同数据类型，例如，**列表**内同时含有**整数**、**浮点数**和**字符串**，甚至一个**列表**中也可以有其他**列表**、**元组**（tuple，第 8 章内容）或是**字典**（dict，第 9 章内容）等当作它的元素，因此，Python 的工作能力，比其他程序语言强大。



列表可以有不同元素，可以用索引取得列表元素内容

6-1-1　列表基本定义

定义列表的语法格式如下：

name_list = [元素 1, … , 元素 n,] # name_list 是假设的列表名称

列表中的每一个数据称为**元素**，这些元素放在中括号 [ ] 内，彼此用逗号 “,” 隔开，上述**元素** n 右边的 “,” 可有可无，这是 Python 设计编译程序人员的贴心设计，因为当元素内容数据量太大时，可能会一行放置一个元素，如下所示：

```
sc = [['洪锦魁', 80, 95, 88, 0],
      ['洪冰儒', 98, 97, 96, 0],
     ]
```

有的设计师处理每个较长的元素时习惯一行放置一个元素，同时习惯在元素末端加上 “,” 符号，处理最后一个元素 n 时有时也习惯加上此逗号，这个概念可以应用在 Python 的其他类似的数据结构上，例如，元组（第 8 章）、字典（第 9 章）、集合（第 10 章）。

如果要打印列表内容，可以用 print( ) 函数，将列表名称当作变量名称即可。

实例 1：NBA 球员 James 前 5 场比赛得分，分别是 23、19、22、31、18，可以用下列方式定义列表。

```
james = [23, 19, 22, 31, 18]
```

实例 2：为所销售的水果——苹果、香蕉、橘子建立列表，可以用下列方式定义列表。

```
fruits = ['apple', 'banana', 'orange']
```

在定义列表时，元素内容也可以使用中文。

实例 3：为所销售的水果——苹果、香蕉、橘子建立中文元素的列表，可以用下列方式定义列表。

```
fruits = ['苹果', '香蕉', '橘子']
```

实例 4：列表内可以有不同的数据类型，例如，在实例 1 的 James 列表最开始的位置增加一个元素，放它的全名。

```
James = ['Lebron James', 23, 19, 22, 31, 18]
```

程序实例 ch6_1.py：定义列表同时打印，最后使用 type( ) 列出列表数据类型。

```
1  # ch6_1.py
2  james = [23, 19, 22, 31, 18]                  # 定义james列表
3  print("打印james列表", james)
4  James = ['Lebron James',23, 19, 22, 31, 18]   # 定义James列表
5  print("打印James列表", James)
6  fruits = ['apple', 'banana', 'orange']        # 定义fruits列表
7  print("打印fruits列表", fruits)
8  cfruits = ['苹果', '香蕉', '橘子']              # 定义cfruits列表
9  print("打印cfruits列表", cfruits)
10 ielts = [5.5, 6.0, 6.5]                       # 定义IELTS成绩列表
11 print("打印IELTS成绩", ielts)
12 # 列出列表数据类型
13 print("列表james数据类型是: ",type(james))
```

执行结果

```
==================== RESTART: D:\Python\ch6\ch6_1.py ====================
打印james列表 [23, 19, 22, 31, 18]
打印James列表 ['Lebron James', 23, 19, 22, 31, 18]
打印fruits列表 ['apple', 'banana', 'orange']
打印cfruits列表 ['苹果', '香蕉', '橘子']
打印IELTS成绩 [5.5, 6.0, 6.5]
列表james数据类型是:  <class 'list'>
```

6-1-2 读取列表元素

可以用列表名称与索引读取列表元素的内容，在 Python 中元素是从索引值 0 开始配置。所以如果是列表的第一个元素，索引值是 0，第二个元素索引值是 1，其他以此类推，如下所示。

```
name_list[i]                        # 读取索引 i 的列表元素
```

程序实例 ch6_2.py：读取列表元素的应用。

```
1  # ch6_2.py
2  james = [23, 19, 22, 31, 18]             # 定义james列表
3  print("打印james第1场得分", james[0])
4  print("打印james第2场得分", james[1])
5  print("打印james第3场得分", james[2])
6  print("打印james第4场得分", james[3])
7  print("打印james第5场得分", james[4])
```

执行结果

```
====================== RESTART: D:\Python\ch6\ch6_2.py ======================
打印james第1场得分 23
打印james第2场得分 19
打印james第3场得分 22
打印james第4场得分 31
打印james第5场得分 18
```

上述程序经过第 2 行的定义后，列表索引值如下。

```
          james[0]  james[2]   james[4]
              ↓         ↓          ↓
james = [23,  19,  22,  31,  18]
                   ↑         ↑
              james[1]  james[3]
```

所以程序第 3 ～ 7 行，可以得到上述执行结果。其实也可以将 2-9 节等号多重指定的概念应用在列表。

程序实例 ch6_3.py：传统处理列表元素内容方式，与 Python 多重指定概念的应用。

```
1  # ch6_3.py
2  james = [23, 19, 22, 31, 18]              # 定义james列表
3  # 传统设计方式
4  game1 = james[0]
5  game2 = james[1]
6  game3 = james[2]
7  game4 = james[3]
8  game5 = james[4]
9  print("打印james各场次得分", game1, game2, game3, game4, game5)
10 # Python高手好的设计方式
11 game1, game2, game3, game4, game5 = james
12 print("打印james各场次得分", game1, game2, game3, game4, game5)
```

执行结果

```
====================== RESTART: D:\Python\ch6\ch6_3.py ======================
打印james各场次得分 23 19 22 31 18
打印james各场次得分 23 19 22 31 18
```

上述程序第 11 行让整个 Python 设计简洁许多，这是 Python 高手常用的程序设计方式，在上述设计中第 11 行的多重指定变量的数量需与列表元素的个数相同，否则会有错误产生。其实懂得用这种方式设计，才算是真正了解 Python 语言的基本精神。

6-1-3　列表切片

在设计程序时，常会需要取得列表的**前几个元素、后几个元素、某区间元素或是依照一定规则排序的元素**，所取得的系列元素也可称为子列表，这个概念称为**列表切片**（list slices），此时可以用下列方法。

```
name_list[start:end]        # 读取从索引 start 到（end-1）的列表元素
name_list[:n]               # 取得列表前 n 名
name_list[:-n]              # 取得列表前面，不含最后 n 名
```

```
name_list[n:]                  # 取得列表索引 n 到最后
name_list[-n:]                 # 取得列表后 n 名
name[:]                        # 取得所有元素，将在 6-8-3 节介绍
```

下列是读取区间，但是用 step 设置每隔多少区间再读取。

```
name_list[start:end:step]      # 每隔 step，读取从索引 start 到 (end-1)
                                 的列表元素
```

实例：列表切片的应用。

```
>>> x = ['0','1','2','3','4','5','6','7','8','9']
>>> x[:3]
['0', '1', '2']
>>> x[:-3]
['0', '1', '2', '3', '4', '5', '6']
>>> x[3:]
['3', '4', '5', '6', '7', '8', '9']
>>> x[-3:]
['7', '8', '9']
```

程序实例 ch6_4.py：列出特定区间球员的得分子列表。

```
1  # ch6_4.py
2  james = [23, 19, 22, 31, 18]                # 定义james列表
3  print("打印james第1-3场得分", james[0:3])
4  print("打印james第2-4场得分", james[1:4])
5  print("打印james第1,3,5场得分", james[0:6:2])
```

执行结果

```
====================== RESTART: D:\Python\ch6\ch6_4.py ======================
打印james第1-3场得分 [23, 19, 22]
打印james第2-4场得分 [19, 22, 31]
打印james第1,3,5场得分 [23, 22, 18]
```

程序实例 ch6_5.py：列出球队前 3 名队员，从索引 1 到最后队员与后 3 名队员子列表。

```
1  # ch6_5.py
2  warriors = ['Curry', 'Durant', 'Iquodala', 'Bell', 'Thompson']
3  first3 = warriors[:3]
4  print("前3名球员",first3)
5  n_to_last = warriors[1:]
6  print("球员索引1到最后",n_to_last)
7  last3 = warriors[-3:]
8  print("后3名球员",last3)
```

执行结果

```
====================== RESTART: D:\Python\ch6\ch6_5.py ======================
前3名球员 ['Curry', 'Durant', 'Iquodala']
球员索引1到最后 ['Durant', 'Iquodala', 'Bell', 'Thompson']
后3名球员 ['Iquodala', 'Bell', 'Thompson']
```

6-1-4　列表索引值是 -1

在列表使用中，如果索引值是 -1，代表是最后一个列表元素。

程序实例 ch6_6.py：列表索引值是 -1 的应用，由下列执行结果可以得到各列表的最后一个元素。

```
1  # ch6_6.py
2  warriors = ['Curry', 'Durant', 'Iquodala', 'Bell', 'Thompson']
3  print("最后一名球员",warriors[-1])
4  james = [23, 19, 22, 31, 18]
5  print("最后一场得分",james[-1])
6  mixs = [9, 20.5, 'DeepStone']
7  print("最后一个元素",mixs[-1])
```

执行结果

```
====================== RESTART: D:\Python\ch6\ch6_6.py ======================
最后一名球员 Thompson
最后一场得分 18
最后一个元素 DeepStone
```

其实在 Python 中索引 -1 代表最后 1 个元素，-2 代表最后第 2 个元素，其他负索引的概念可以此类推，可参考下列实例。

程序实例 ch6_7.py：使用负索引列出 warriors 列表内容。

```
1  # ch6_7.py
2  warriors = ['Curry', 'Durant', 'Iquodala', 'Bell', 'Thompson']
3  print(warriors[-1],warriors[-2],warriors[-3],warriors[-4],warriors[-5])
```

执行结果

```
====================== RESTART: D:/Python/ch6/ch6_7.py ======================
Thompson Bell Iquodala Durant Curry
```

6-1-5　列表最大值 max( )、最小值 min( )、总和 sum( )

Python 内建了一些执行统计运算的函数，如果列表内容全部是数值则可以使用 max( ) 函数获得列表的最大值，min( ) 函数可以获得列表的最小值，sum( ) 函数可以获得列表的总和。如果列表内容全部是字符或字符串，则可以使用 max( ) 函数获得列表的 Unicode 码值的最大值，min( ) 函数可以获得列表的 Unicode 码值最小值。sum( ) 则不可使用在列表元素为非数值的情况。

程序实例 ch6_8.py：计算 James 球员 5 场的最高得分、最少得分和 5 场的得分总计。

```
1  # ch6_8.py
2  james = [23, 19, 22, 31, 18]                # 定义James的5场比赛得分
3  print("最高得分 = ", max(james))
4  print("最低得分 = ", min(james))
5  print("得分总计 = ", sum(james))
```

执行结果

```
====================== RESTART: D:\Python\ch6\ch6_8.py ======================
最高得分 =  31
最低得分 =  18
得分总计 =  113
```

上述程序很快地获得了统计信息，读者可能会想，如果在列表内含有字符串，例如，程序实例 ch6_1.py 的 James 列表，这个列表索引 0 元素是字符串，如果这时仍然直接用 max（James）会有错误的。

```
>>> James = ['Lebron James', 23, 19, 22, 31, 18]
>>> x = max(James)
Traceback (most recent call last):
  File "<pyshell#83>", line 1, in <module>
    x = max(James)
TypeError: '>' not supported between instances of 'int' and 'str'
>>>
```

碰上这类字符串可以使用 6-1-3 节中的方式，用切片方式处理，如下所示。

程序实例 ch6_9.py：重新设计 ch6_8.py，但是使用含字符串元素的 James 列表。

```
1  # ch6_9.py
2  James = ['Lebron James', 23, 19, 22, 31, 18] # 定义James的5场比赛得分
3  print("最高得分 = ", max(James[1:6]))
4  print("最低得分 = ", min(James[1:6]))
5  print("得分总计 = ", sum(James[1:6]))
```

执行结果

```
====================== RESTART: D:\Python\ch6\ch6_9.py ======================
最高得分 =  31
最低得分 =  18
得分总计 =  113
```

6-1-6 列表个数 len( )

设计程序时，可能会增加元素，也有可能会删除元素，时间久了即使是程序设计师也无法得知列表内剩余多少元素，此时可以借用本节的 len( ) 函数获得列表的元素个数。

程序实例 ch6_10.py：重新设计 ch6_8.py，获得场次数据。

```
1  # ch6_10.py
2  james = [23, 19, 22, 31, 18]          # 定义James的5场比赛得分
3  games = len(james)                    # 获得场次数据
4  print("经过 %d 场比赛最高得分 = " % games, max(james))
5  print("经过 %d 场比赛最低得分 = " % games, min(james))
6  print("经过 %d 场比赛得分总计 = " % games, sum(james))
```

执行结果

```
====================== RESTART: D:\Python\ch6\ch6_10.py ======================
经过 5 场比赛最高得分 =  31
经过 5 场比赛最低得分 =  18
经过 5 场比赛得分总计 =  113
>>>
```

6-1-7 更改列表元素的内容

可以使用列表名称和索引值更改列表元素的内容。

程序实例 ch6_11.py：修改 James 第 5 场比赛分数。

```
1  # ch6_11.py
2  james = [23, 19, 22, 31, 18]       # 定义James的5场比赛得分
3  print("旧的James比赛分数", james)
4  james[4] = 28
5  print("新的James比赛分数", james)
```

执行结果

```
=============== RESTART: D:\Python\ch6\ch6_11.py ===============
s比赛分数 [23, 19, 22, 31, 18]
s比赛分数 [23, 19, 22, 31, 28]
```

这个概念可以用于更改整数数据，也可以修改字符串数据。

程序实例 ch6_12.py：一家汽车经销商原本可以销售 Toyota、Nissan、Honda，现在 Nissan 销售权被回收，改成销售 Ford，可用下列方式设计销售品牌。

```
1  # ch6_12.py
2  cars = ['Toyota', 'Nissan', 'Honda']
3  print("旧汽车销售品牌", cars)
4  cars[1] = 'Ford'              # 更改第二个元素内容
5  print("新汽车销售品牌", cars)
```

执行结果

```
====================== RESTART: D:\Python\ch6\ch6_12.py ======================
旧汽车销售品牌 ['Toyota', 'Nissan', 'Honda']
新汽车销售品牌 ['Toyota', 'Ford', 'Honda']
```

6-1-8　列表的相加

Python 是允许列表相加的，相当于将列表结合。

程序实例 ch6_13.py：一家汽车经销商原本可以销售 Toyota、Nissan、Honda，现在并购一家销售 Audi、BMW 的经销商，可用下列方式设计销售品牌。

```
1  # ch6_13.py
2  cars1 = ['Toyota', 'Nissan', 'Honda']
3  print("旧汽车销售品牌", cars1)
4  cars2 = ['Audi', 'BMW']
5  cars1 += cars2
6  print("新汽车销售品牌", cars1)
```

执行结果

```
====================== RESTART: D:\Python\ch6\ch6_13.py ======================
旧汽车销售品牌 ['Toyota', 'Nissan', 'Honda']
新汽车销售品牌 ['Toyota', 'Nissan', 'Honda', 'Audi', 'BMW']
```

程序实例 ch6_14.py：整数列表相加的实例。

```
1  # ch6_14.py
2  num1 = [1, 3, 5]
3  num2 = [2, 4, 6]
4  num3 = num1 + num2            # 字符串为主的列表相加
5  print(num3)
```

执行结果

```
===================== RESTART: D:/Python/ch6/ch6_14.py =====================
[1, 3, 5, 2, 4, 6]
```

6-1-9 列表乘以一个数字

如果将列表乘以一个数字，这个数字相当于是列表元素重复次数。

程序实例 ch6_15.py：将列表乘以数字的应用。

```
1  # ch6_15.py
2  cars = ['toyota', 'nissan', 'honda']
3  nums = [1, 3, 5]
4  carslist = cars * 3            # 列表乘以数字
5  print(carslist)
6  numslist = nums * 5            # 列表乘以数字
7  print(numslist)
```

执行结果

```
===================== RESTART: D:/Python/ch6/ch6_15.py =====================
['toyota', 'nissan', 'honda', 'toyota', 'nissan', 'honda', 'toyota', 'nissan',
 'honda']
[1, 3, 5, 1, 3, 5, 1, 3, 5, 1, 3, 5, 1, 3, 5]
```

注 Python 的列表**不支持**列表加上数字，例如，第 6 行改成如下所示：

```
numslist = nums + 5            # 列表加上数字将造成错误
```

6-1-10 列表元素的加法操作

既然可以读取列表内容，其实就可以使用相同的概念操作列表内的元素数据。

程序实例 ch6_16.py：建立 Lebron James 和 Kevin Love 在比赛中的得分列表，然后利用列表元素加法操作，列出两个人在第四场比赛的得分总和。

```
1  # ch6_16.py
2  James = ['Lebron James',23, 19, 22, 31, 18] # 定义James列表
3  Love = ['Kevin Love',20, 18, 30, 22, 15]    # 定义Love列表
4  game3 = James[4] + Love[4]
5  LKgame = James[0] + ' 和 ' + Love[0] + '第四场总得分 = '
6  print(LKgame, game3)
```

执行结果

```
===================== RESTART: D:\Python\ch6\ch6_16.py =====================
Lebron James 和 Kevin Love第四场总得分 =  53
```

需注意由第 2 行列表定义可知，James[0] 是指“Lebron James”，James[1] 是第 1 场得分 23，所以 James[4] 是第 4 场得分 31。第 3 行 Love 列表含义相同。上述第 5 行是整数和字符串相加，相当于产生新字符串。

6-1-11 删除列表元素

可以使用下列方式删除指定索引的列表元素。

del name_list[i] # 删除索引 i 的列表元素

下列是删除列表区间元素。

del name_list[start:end] # 删除从索引 start 到 (end-1) 的列表元素

下列是删除区间，但是用 step 设置每隔多少区间再删除。

del name_list[start:end:step] # 每隔 step，删除从索引 start 到 (end-1) 的列表元素

程序实例 ch6_17.py：如果 NBA 勇士队主将阵容有 5 名，其中一名队员 Bell 离队了，可用下列方式设计。

```
1  # ch6_17.py
2  warriors = ['Curry', 'Durant', 'Iquodala', 'Bell', 'Thompson']
3  print("2018年初NBA勇士队主将阵容", warriors)
4  del warriors[3]                    # 不明原因离队
5  print("2018年末NBA勇士队主将阵容", warriors)
```

执行结果

```
====================== RESTART: D:\Python\ch6\ch6_17.py ======================
2018年初NBA勇士队主将阵容 ['Curry', 'Durant', 'Iquodala', 'Bell', 'Thompson']
2018年末NBA勇士队主将阵容 ['Curry', 'Durant', 'Iquodala', 'Thompson']
```

程序实例 ch6_18.py：删除列表元素的应用。

```
1  # ch6_18.py
2  nums1 = [1, 3, 5]
3  print("删除nums1列表索引1元素前    = ",nums1)
4  del nums1[1]
5  print("删除nums1列表索引1元素后    = ",nums1)
6  nums2 = [1, 2, 3, 4, 5, 6]
7  print("删除nums2列表索引[0:2]前    = ",nums2)
8  del nums2[0:2]
9  print("删除nums2列表索引[0:2]后    = ",nums2)
10 nums3 = [1, 2, 3, 4, 5, 6]
11 print("删除nums3列表索引[0:6:2]前 = ",nums3)
12 del nums3[0:6:2]
13 print("删除nums3列表索引[0:6:2]后 = ",nums3)
```

执行结果

```
====================== RESTART: D:\Python\ch6\ch6_18.py ======================
删除nums1列表索引1元素前    =  [1, 3, 5]
删除nums1列表索引1元素后    =  [1, 5]
删除nums2列表索引[0:2]前    =  [1, 2, 3, 4, 5, 6]
删除nums2列表索引[0:2]后    =  [3, 4, 5, 6]
删除nums3列表索引[0:6:2]前 =  [1, 2, 3, 4, 5, 6]
删除nums3列表索引[0:6:2]后 =  [2, 4, 6]
```

以这种方式删除列表元素最大的缺点是，删除元素后无法得知删除的是什么内容。有时在设计网站时，可能想将某个人从 VIP 客户降为一般客户，采用上述方式删除元素时，就无法再度取得所

删除的元素数据，6-4-3 节会介绍另一种删除数据方式，删除后还可善加利用所删除的数据。又或者你设计一个游戏，敌人是放在列表内，采用上述方式删除所杀死的敌人时，就无法再度取得所删除的敌人元素数据，如果可以取得的话，可以在杀死敌人坐标位置放置庆祝动画等。

6-1-12　列表为空列表的判断

如果想建立一个列表，可是暂时不放置元素，可使用下列方式声明。

```
name_list = [ ]                    # 这是空的列表
```

程序实例 ch6_19.py：删除列表元素的应用，这个程序基本上会用 len( ) 函数判断列表内是否有元素数据，如果有则删除索引为 0 的元素，如果没有则列出列表内没有元素了。

```
1  # ch6_19.py
2  cars = ['Toyota', 'Nissan', 'Honda']
3  print("cars列表长度是 = %d" %  len(cars))
4  if len(cars) != 0:
5      del cars[0]
6      print("删除cars列表元素成功")
7      print("cars列表长度是 = %d" % len(cars))
8  else:
9      print("cars列表内没有元素数据")
10 nums = []
11 print("nums列表长度是 = %d" % len(nums))
12 if len(nums) != 0:
13     del nums[0]
14     print("删除nums列表元素成功")
15 else:
16     print("nums列表内没有元素数据")
```

执行结果

```
===================== RESTART: D:\Python\ch6\ch6_19.py =====================
cars列表长度是 = 3
删除cars列表元素成功
cars列表长度是 = 2
nums列表长度是 = 0
nums列表内没有元素数据
```

6-1-13　删除列表

Python 也允许删除整个列表，列表一经删除后就无法复原，同时也无法做任何操作了，下面是删除列表的方式。

```
del name_list                      # 删除列表 name_list
```

实例：建立列表、打印列表、删除列表，然后尝试再度打印列表结果出现错误消息，因为列表经删除后已经不存在了。

```
>>> x = [1,2,3]
>>> print(x)
[1, 2, 3]
>>> del x
>>> print(x)
Traceback (most recent call last):
  File "<pyshell#25>", line 1, in <module>
    print(x)
NameError: name 'x' is not defined
>>>
```

6-1-14　补充多重指定与列表

在多重指定中，如果等号左边的变量较少，可以用“* 变量”方式，将多余的右边内容用列表方式打包给含“*”的变量。

实例 1：将多的内容打包给 c。

```
>>> a, b, *c = 1, 2, 3, 4, 5
>>> print(a, b, c)
1 2 [3, 4, 5]
```

变量内容打包时，不一定要在最右边，可以在任意位置。

实例 2：将多的内容打包给 b。

```
>>> a, *b, c = 1, 2, 3, 4, 5
>>> print(a, b, c)
1 [2, 3, 4] 5
```

6-2 Python 简单的面向对象概念

在面向对象的程序设计里，所有数据都算是一个**对象**（Object），例如，**整数**、**浮点数**、**字符串**或是本章所提的**列表**都是一个对象。我们可以为所建立的对象设计一些**方法**（method），供这些对象使用，在这里所提的**方法**表面是**函数**，但是这些函数是放在类（第 12 章会介绍）内，我们称之为**方法**，它与函数调用方式不同。目前 Python 有为一些基本对象提供默认的**方法**，要使用这些**方法**可以在**对象**后先放**小数点**，再放**方法名称**，基本语法格式如下：

对象 . 方法 ( )

下面是字符串常用的方法。

lower ( )：将字符串转成小写字母。(6-2-1 节)

upper ( )：将字符串转成大写字母。(6-2-1 节)

title ( )：将字符串转成第一个字母大写，其他字母小写。(6-2-1 节)

rstrip ( )：删除字符串尾端多余的空白。(6-2-2 节)

lstrip ( )：删除字符串开始端多余的空白。(6-2-2 节)

strip ( )：删除字符串头尾两边多余的空白。(6-2-2 节)

center ( )：字符串在指定宽度居中对齐。(6-2-3 节)

rjust ( )：字符串在指定宽度靠右对齐。(6-2-3 节)

ljust ( )：字符串在指定宽度靠左对齐。(6-2-3 节)

下面将分成几节一步一步以实例进行说明。

6-2-1　更改字符串大小写 lower( )/upper( )/title( )

如果列表内的元素字符串数据是小写，例如，输出的车辆名称是 "benz"，可以使用 title( ) 让车辆名称的第一个字母大写，可能会更好。

程序实例 ch6_20.py：将 upper( ) 和 title( ) 应用于字符串。

```
1  # ch6_20.py
2  cars = ['bmw', 'benz', 'audi']
3  carF = "我开的第一部车是 " + cars[1].title( )
4  carN = "我现在开的车子是 " + cars[0].upper( )
5  print(carF)
6  print(carN)
```

执行结果

```
====================== RESTART: D:\Python\ch6\ch6_20.py ======================
我开的第一部车是 Benz
我现在开的车子是 BMW
```

上述第 3 行是将 benz 改为 Benz，第 4 行是将 bmw 改为 BMW。下列是使用 lower( ) 将字符串改为小写的实例。

```
>>> x = 'ABC'
>>> x.lower()
'abc'
```

使用 title( ) 时需留意，如果字符串内含多个单词，所有的单词均是第一个字母大写。

```
>>> x = "i love python"
>>> x.title()
'I Love Python'
```

6-2-2 删除空格符 rstrip( )/lstrip( )/strip( )

删除字符串开始或结尾多余空格是很好用的方法（method），特别是系统要求读者输入数据时，一定会有人不小心多输入一些空格符，此时可以用这个方法删除多余的空格。

程序实例 ch6_21.py：删除开始端与结尾端多余空格的应用。

```
1  # ch6_21.py
2  strN = " DeepStone "
3  strL = strN.lstrip( )          # 删除字符串左边多余空格
4  strR = strN.rstrip( )          # 删除字符串右边多余空格
5  strB = strN.lstrip( )          # 先删除字符串左边多余空格
6  strB = strB.rstrip( )          # 再删除字符串右边多余空格
7  strO = strN.strip( )           # 一次删除头尾端多余空格
8  print("/%s/" % strN)
9  print("/%s/" % strL)
10 print("/%s/" % strR)
11 print("/%s/" % strB)
12 print("/%s/" % strO)
```

执行结果

```
====================== RESTART: D:/Python/ch6/ch6_21.py ======================
/ DeepStone /
/DeepStone /
/ DeepStone/
/DeepStone/
/DeepStone/
```

删除前后空格符常常应用于读取屏幕输入，除了上述，下面将用实例说明整个影响。

程序实例 ch6_22.py：没有使用 strip( ) 与使用 strip( ) 方法处理读取字符串的观察。

```
1  # ch6_22.py
2  string = input("请输入名字 : ")
3  print("/%s/" % string)
4  string = input("请输入名字 : ")
5  print("/%s/" % string.strip())
```

执行结果 下列是第一个数据的输入，同时不使用 strip( ) 方法。

```
====================== RESTART: D:/Python/ch6/ch6_22.py ======================
请输入名字 :          DeepStone     |
```

插入点

按 Enter 键后可以得到下列输出。

```
====================== RESTART: D:/Python/ch6/ch6_22.py ======================
请输入名字:          DeepStone
/         DeepStone         /
请输入名字: |
```

下列是第 2 个数据的输入，使用了 strip( ) 方法。

```
====================== RESTART: D:/Python/ch6/ch6_22.py ======================
请输入名字 :          DeepStone
/         DeepStone         /
请输入名字 :          DeepStone     |
```

插入点

按 Enter 键后可以得到下列输出。

```
====================== RESTART: D:/Python/ch6/ch6_22.py ======================
请输入名字 :          DeepStone
/         DeepStone         /
请输入名字 :          DeepStone
/DeepStone/
```

6-2-3　格式化字符串位置 center( )/ljust( )/rjust( )

这几个函数用于格式化字符串，可以给出一定的字符串长度空间，然后可以看到字符串分别居中（center)、靠左（ljust)、靠右 rjust( ) 对齐。

程序实例 ch6_23.py：格式化字符串位置的应用。

```
1  # ch6_23.py
2  title = "Ming-Chi Institute of Technology"
3  print("/%s/" % title.center(50))
4  dt = "Department of ME"
5  print("/%s/" % dt.ljust(50))
6  site = "JK Hung"
7  print("/%s/" % site.rjust(50))
```

执行结果

```
====================== RESTART: D:\Python\ch6\ch6_23.py ======================
/         Ming-Chi Institute of Technology         /
/Department of ME                                  /
/                                           JK Hung/
```

如果发生预留空间不足时，Python 也会配置足够的空间。

6-2-4 dir( ) 获得系统内部对象的方法

6-2 节列举了字符串常用的**方法**（method），dir( ) 函数可以列出对象有哪些内建的方法可以使用。

实例 1：列出字符串对象的方法，处理方式是可以先设置一个字符串变量，再将此字符串变量当作 dir( ) 的参数，最后列出此字符串变量的方法（method）。

```
>>> string = 'abc'
>>> dir(string)
['__add__', '__class__', '__contains__', '__delattr__', '__dir__', '__doc__',
'__eq__', '__format__', '__ge__', '__getattribute__', '__getitem__', '__getnew
args__', '__gt__', '__hash__', '__init__', '__init_subclass__', '__iter__', '_
_le__', '__len__', '__lt__', '__mod__', '__mul__', '__ne__', '__new__', '__red
uce__', '__reduce_ex__', '__repr__', '__rmod__', '__rmul__', '__setattr__', '_
_sizeof__', '__str__', '__subclasshook__', 'capitalize', 'casefold', 'center',
'count', 'encode', 'endswith', 'expandtabs', 'find', 'format', 'format_map',
'index', 'isalnum', 'isalpha', 'isascii', 'isdecimal', 'isdigit', 'isidentifie
r', 'islower', 'isnumeric', 'isprintable', 'isspace', 'istitle', 'isupper', 'j
oin', 'ljust', 'lower', 'lstrip', 'maketrans', 'partition', 'replace', 'rfind'
, 'rindex', 'rjust', 'rpartition', 'rsplit', 'rstrip', 'split', 'splitlines',
'startswith', 'strip', 'swapcase', 'title', 'translate', 'upper', 'zfill']
```

其实上述语句设置了 string='abc'，Python 内部已经建立了一个数据结构供变量 string 使用，同时设置了内容是字符串 'abc'，接着 Python 将数据结构调整为字符串数据结构，所以使用 dir(string) 时，会列出适用字符串使用的方法。

上述有圈起来的，在前几节已有介绍。看到上述密密麻麻的方法，读者不用紧张，也不用想要一次学会，需要时再学即可。如果想要了解上述特定方法，可以使用 4-1 节介绍的 help( ) 函数，可以用下列方式：

help(对象 . 方法名称)

实例 2：延续前一个实例，列出对象 string 内建的 islower 的使用说明，同时以 string 对象为例，测试使用结果。

```
>>> help(string.islower)
Help on built-in function islower:

islower(...) method of builtins.str instance
    S.islower() -> bool

    Return True if all cased characters in S are lowercase and there is
    at least one cased character in S, False otherwise.

>>> x = string.islower()
>>> print(x)
True
>>>
```

由上述说明可知，islower( ) 可以返回对象是否是小写，如果对象全部是小写或是至少有一个字符是小写将返回 True，否则返回 False。在上述实例中，由于 string 对象的内容是 "abc"，全部是小写，所以返回 True。

上述概念同样可以应用在查询整数对象的方法。

实例 3：列出整数对象的方法，同样可以先设置一个整数变量，再列出此整数变量的方法（method）。

```
>>> num = 5
>>> dir(num)
['__abs__', '__add__', '__and__', '__bool__', '__ceil__', '__class__', '__delattr__',
'__dir__', '__divmod__', '__doc__', '__eq__', '__float__', '__floor__', '__floordiv__
', '__format__', '__ge__', '__getattribute__', '__getnewargs__', '__gt__', '__hash__'
, '__index__', '__init__', '__init_subclass__', '__int__', '__invert__', '__le__', '_
_lshift__', '__lt__', '__mod__', '__mul__', '__ne__', '__neg__', '__new__', '__or__',
'__pos__', '__pow__', '__radd__', '__rand__', '__rdivmod__', '__reduce__', '__reduce_
ex__', '__repr__', '__rfloordiv__', '__rlshift__', '__rmod__', '__rmul__', '__ror__',
'__round__', '__rpow__', '__rrshift__', '__rshift__', '__rsub__', '__rtruediv__', '__
rxor__', '__setattr__', '__sizeof__', '__str__', '__sub__', '__subclasshook__', '__tr
uediv__', '__trunc__', '__xor__', 'bit_length', 'conjugate', 'denominator', 'from_byt
es', 'imag', 'numerator', 'real', 'to_bytes']
>>>
```

上述 bit_length 是可以计算出要多少位以二进制方式存储此变量。

实例 4：列出需要多少位，存储整数变量 num。

```
>>> num = 5
>>> y = num.bit_length()
>>> y
3
>>> num = 31
>>> y = num.bit_length()
>>> y
5
```

6-3 获得列表的方法

本节重点是列表，可以使用下列方式获得可以使用哪些列表的方法。

实例 1：列出内建列表（list）内含字符串（string）元素的方法。

```
>>> string = ["bmw", "benz", "audi"]
>>> dir(string)
['__add__', '__class__', '__contains__', '__delattr__', '__delitem__', '__dir__', '__
doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__getitem__', '__gt__'
, '__hash__', '__iadd__', '__imul__', '__init__', '__init_subclass__', '__iter__', '_
_le__', '__len__', '__lt__', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_
ex__', '__repr__', '__reversed__', '__rmul__', '__setattr__', '__setitem__', '__sizeo
f__', '__str__', '__subclasshook__', 'append', 'clear', 'copy', 'count', 'extend', 'i
ndex', 'insert', 'pop', 'remove', 'reverse', 'sort']
>>>
```

上述实例的重点是先建立一个列表 string，然后由此列表利用 dir( ) 函数可以了解有哪些列表的方法可以使用。

实例 2：列出内建列表（list）内含整数（int）元素的方法。

```
>>> numlist = [1, 3, 5]
>>> dir(numlist)
['__add__', '__class__', '__contains__', '__delattr__', '__delitem__', '__dir__', '__
doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__getitem__', '__gt__'
, '__hash__', '__iadd__', '__imul__', '__init__', '__init_subclass__', '__iter__', '_
_le__', '__len__', '__lt__', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_
ex__', '__repr__', '__reversed__', '__rmul__', '__setattr__', '__setitem__', '__sizeo
f__', '__str__', '__subclasshook__', 'append', 'clear', 'copy', 'count', 'extend', 'i
ndex', 'insert', 'pop', 'remove', 'reverse', 'sort']
>>>
```

可以看到实例 1 与实例 2 内容完全相同，这表示下一节起讲解操作列表的方法，可以用于元素是字符串，也可以用于元素是整数。

6-4 增加与删除列表元素

6-4-1 在列表末端增加元素 append( )

程序设计时常常会发生需要增加列表元素的情况，如果目前元素个数是 3 个，想要增加第 4 个元素，读者可能会想可否使用下列传统方式，直接设置新增的值：

```
name_list[3] = value
```

实例：使用索引方式，为列表增加元素，但是发生索引值超过列表长度的错误。

```
>>> car = ['Honda', 'Toyota', 'Ford']
>>> print(car)
['Honda', 'Toyota', 'Ford']
>>> car[3] = 'Nissan'
Traceback (most recent call last):
  File "<pyshell#31>", line 1, in <module>
    car[3] = 'Nissan'
IndexError: list assignment index out of range
>>>
```

读者可能会想可以增加一个新列表，将要新增的元素放在新列表，然后再将原列表与新列表相加，就达到增加列表元素的目的了。这个方法理论上是可以，可是太麻烦了。Python 为列表内建了新增元素的方法 append( )，这个方法可以在列表末端直接增加元素。

```
name_list.append('新增元素')
```

程序实例 ch6_24.py：先建立一个空列表，然后分别使用 append( ) 增加 3 个元素内容。

```
1  # ch6_24.py
2  cars = []
3  print("目前列表内容 = ",cars)
4  cars.append('Honda')
5  print("目前列表内容 = ",cars)
6  cars.append('Toyota')
7  print("目前列表内容 = ",cars)
8  cars.append('Ford')
9  print("目前列表内容 = ",cars)
```

执行结果

```
===================== RESTART: D:\Python\ch6\ch6_24.py =====================
目前列表内容 =  []
目前列表内容 =  ['Honda']
目前列表内容 =  ['Honda', 'Toyota']
目前列表内容 =  ['Honda', 'Toyota', 'Ford']
```

6-4-2 插入列表元素 insert( )

append( ) 方法是固定在列表末端插入元素，insert( ) 方法则是可以在任意位置插入元素，它的使用格式如下：

```
insert(索引, 元素内容)          # 索引是插入位置，元素内容是插入内容
```

程序实例 ch6_25.py：使用 insert( ) 插入列表元素的应用。

```
1  # ch6_25.py
2  cars = ['Honda','Toyota','Ford']
3  print("目前列表内容 = ",cars)
4  print("在索引1位置插入Nissan")
5  cars.insert(1,'Nissan')
6  print("新的列表内容 = ",cars)
7  print("在索引0位置插入BMW")
8  cars.insert(0,'BMW')
9  print("最新列表内容 = ",cars)
```

执行结果

```
===================== RESTART: D:\Python\ch6\ch6_25.py =====================
目前列表内容 =  ['Honda', 'Toyota', 'Ford']
在索引1位置插入Nissan
新的列表内容 =  ['Honda', 'Nissan', 'Toyota', 'Ford']
在索引0位置插入BMW
最新列表内容 =  ['BMW', 'Honda', 'Nissan', 'Toyota', 'Ford']
```

6-4-3　删除列表元素 pop( )

6-1-11 节介绍了使用 del 删除列表元素，同时指出这种方法最大的缺点是，资料删除了就无法取得相关信息。使用 pop( ) 方法删除元素最大的优点是，删除后将返回所删除的值。使用 pop( ) 时若是未指明所删除元素的位置，一律删除列表末端的元素。pop( ) 的使用方式如下。

```
value = name_list.pop( )            # 没有索引时删除列表末端元素
value = name_list.pop(i)            # 删除指定索引值 i 位置的列表元素
```

程序实例 ch6_26.py：使用 pop( ) 删除列表元素的应用，这个程序第 5 行未指明删除的索引值，所以删除了列表的最后一个元素。程序第 9 行则是指明删除索引为 1 位置的元素。

```
1  # ch6_26.py
2  cars = ['Honda','Toyota','Ford','BMW']
3  print("目前列表内容 = ",cars)
4  print("使用pop( )删除列表元素")
5  popped_car = cars.pop( )          # 删除列表末端值
6  print("所删除的列表内容是 : ", popped_car)
7  print("新的列表内容 = ",cars)
8  print("使用pop(1)删除列表元素")
9  popped_car = cars.pop(1)          # 删除列表索引为1的值
10 print("所删除的列表内容是 : ", popped_car)
11 print("新的列表内容 = ",cars)
```

执行结果

```
===================== RESTART: D:\Python\ch6\ch6_26.py =====================
目前列表内容 =  ['Honda', 'Toyota', 'Ford', 'BMW']
使用pop( )删除列表元素
所删除的列表内容是 :  BMW
新的列表内容 =  ['Honda', 'Toyota', 'Ford']
使用pop(1)删除列表元素
所删除的列表内容是 :  Toyota
新的列表内容 =  ['Honda', 'Ford']
```

6-4-4　删除指定的元素 remove( )

在删除列表元素时，有时可能不知道元素在列表内的位置，此时可以使用 remove( ) 方法删除

指定的元素，它的使用方式如下：

name_list.remove(想删除的元素内容)

如果列表内有相同的元素，则只删除第一个出现的元素，如果想要删除所有相同的元素，必须使用循环，第 7 章将会讲解循环的概念。

程序实例 ch6_27.py：删除列表中第一次出现的元素 bmw，这个列表有两个 bmw 字符串，最后只删除索引为 1 位置的 bmw 字符串。

```
1  # ch6_27.py
2  cars = ['Honda','bmw','Toyota','Ford','bmw']
3  print("目前列表内容 = ",cars)
4  print("使用remove( )删除列表元素")
5  expensive = 'bmw'
6  cars.remove(expensive)              # 删除第一次出现的元素bmw
7  print("所删除的内容是: " + expensive.upper( ) + " 因为太贵了" )
8  print("新的列表内容",cars)
```

执行结果

```
====================== RESTART: D:\Python\ch6\ch6_27.py ======================
目前列表内容 =  ['Honda', 'bmw', 'Toyota', 'Ford', 'bmw']
使用remove()删除列表元素
所删除的内容是: BMW 因为太贵了
新的列表内容 ['Honda', 'Toyota', 'Ford', 'bmw']
```

6-5 列表的排序

6-5-1 颠倒排序 reverse( )

reverse( ) 可以颠倒排序列表元素，它的使用方式如下：

name_list.reverse( )   # 颠倒排序 name_list 列表元素

列表经颠倒排放后，就算永久性更改了，如果要复原，可以再执行一次 reverse( ) 方法。

其实在 6-1-3 节的切片应用中，也可以用 [::-1] 方式取得列表颠倒排序，这个方式会返回新的颠倒排序列表，原列表顺序未改变。

程序实例 ch6_28.py：使用两种方式执行颠倒排序列表元素。

```
1  # ch6_28.py
2  cars = ['Honda','bmw','Toyota','Ford','bmw']
3  print("目前列表内容 = ",cars)
4  # 直接打印cars[::-1]颠倒排序,不更改列表内容
5  print("打印使用[::-1]颠倒排序\n", cars[::-1])
6  # 更改列表内容
7  print("使用reverse()颠倒排序列表元素")
8  cars.reverse()                  # 颠倒排序列表
9  print("新的列表内容 = ",cars)
```

执行结果

```
======================= RESTART: D:\Python\ch6\ch6_28.py =======================
目前列表内容 =  ['Honda', 'bmw', 'Toyota', 'Ford', 'bmw']
打印使用[::-1]颠倒排序
 ['bmw', 'Ford', 'Toyota', 'bmw', 'Honda']
使用reverse()颠倒排序列表元素
新的列表内容 =  ['bmw', 'Ford', 'Toyota', 'bmw', 'Honda']
```

6-5-2　sort( ) 排序

sort( ) 方法可以对列表元素由小到大排序，这个方法对纯数值元素与纯英文字符串元素都有非常好的效果。要留意的是，经排序后原列表的元素顺序会被永久更改。它的使用格式如下：

```
name_list.sort( )                    # 由小到大排序 name_list 列表
```

如果是排序英文字符串，建议先将字符串英文字符全部改成小写或全部改成大写。

程序实例 ch6_29.py：数字与英文字符串元素排序的应用。

```
1  # ch6_29.py
2  cars = ['honda','bmw','toyota','ford']
3  print("目前列表内容 = ",cars)
4  print("使用sort( )由小排到大")
5  cars.sort( )
6  print("排序列表结果 = ",cars)
7  nums = [5, 3, 9, 2]
8  print("目前列表内容 = ",nums)
9  print("使用sort( )由小排到大")
10 nums.sort()
11 print("排序列表结果 = ",nums)
```

执行结果

```
======================= RESTART: D:\Python\ch6\ch6_29.py =======================
目前列表内容 =  ['honda', 'bmw', 'toyota', 'ford']
使用sort( )由小排到大
排序列表结果 =  ['bmw', 'ford', 'honda', 'toyota']
目前列表内容 =  [5, 3, 9, 2]
使用sort( )由小排到大
排序列表结果 =  [2, 3, 5, 9]
```

上述内容是由小排到大，sort( ) 方法也允许由大排到小，只要在 sort( ) 内增加参数“reverse=True”即可。

程序实例 ch6_30.py：重新设计 ch6_29.py，将列表元素由大排到小。

```
1  # ch6_30.py
2  cars = ['honda','bmw','toyota','ford']
3  print("目前列表内容 = ",cars)
4  print("使用sort( )由大排到小")
5  cars.sort(reverse=True)
6  print("排序列表结果 = ",cars)
7  nums = [5, 3, 9, 2]
8  print("目前列表内容 = ",nums)
9  print("使用sort( )由大排到小")
10 nums.sort(reverse=True)
11 print("排序列表结果 = ",nums)
```

执行结果

```
====================== RESTART: D:\Python\ch6\ch6_30.py ======================
目前列表内容 =  ['honda', 'bmw', 'toyota', 'ford']
使用sort( )由大排到小
排序列表结果 =  ['toyota', 'honda', 'ford', 'bmw']
目前列表内容 =  [5, 3, 9, 2]
使用sort()由大排到小
排序列表结果 =  [9, 5, 3, 2]
```

6-5-3 sorted( ) 排序

6-5-2 节的 sort( ) 排序将造成列表元素顺序永久更改，如果不希望更改列表元素顺序，可以使用另一种排序 sorted( )，使用这个排序可以获得想要的排序结果。可以用新列表存储新的排序列表，同时原列表的顺序将不更改。它的使用格式如下：

```
new_list = sorted(name_list)          # 用新列表存储排序，原列表序列不更改
```

程序实例 ch6_31.py：sorted( ) 排序的应用，这个程序使用 car_sorted 新列表存储 car 列表的排序结果，同时使用 num_sorted 新列表存储 num 列表的排序结果。

```
1  # ch6_31.py
2  cars = ['honda','bmw','toyota','ford']
3  print("目前列表car内容 = ",cars)
4  print("使用sorted()由小排到大")
5  cars_sorted = sorted(cars)
6  print("排序列表结果 = ",cars_sorted)
7  print("原先列表car内容 = ",cars)
8  nums = [5, 3, 9, 2]
9  print("目前列表num内容 = ",nums)
10 print("使用sorted()由小排到大")
11 nums_sorted = sorted(nums)
12 print("排序列表结果 = ",nums_sorted)
13 print("原先列表num内容 = ",nums)
```

执行结果

```
====================== RESTART: D:\Python\ch6\ch6_31.py ======================
目前列表car内容 =  ['honda', 'bmw', 'toyota', 'ford']
使用sorted( )由小排到大
排序列表结果 =  ['bmw', 'ford', 'honda', 'toyota']
原先列表car内容 =  ['honda', 'bmw', 'toyota', 'ford']
目前列表num内容 =  [5, 3, 9, 2]
使用sorted()由小排到大
排序列表结果 =  [2, 3, 5, 9]
原先列表num内容 =  [5, 3, 9, 2]
```

如果想要从大排到小，可以在 sorted( ) 内增加参数“reverse=True”，可参考下列实例第 5 和 11 行。

程序实例 ch6_32.py：重新设计 ch6_31.py，将列表由大排到小。

```
1  # ch6_32.py
2  cars = ['honda','bmw','toyota','ford']
3  print("目前列表car内容 = ",cars)
4  print("使用sorted()由大排到小")
5  cars_sorted = sorted(cars,reverse=True)
6  print("排序列表结果     = ",cars_sorted)
7  print("原先列表car内容 = ",cars)
8  nums = [5, 3, 9, 2]
9  print("目前列表num内容 = ",nums)
10 print("使用sorted()由大排到小")
11 nums_sorted = sorted(nums,reverse=True)
12 print("排序列表结果     = ",nums_sorted)
13 print("原先列表num内容 = ",nums)
```

执行结果

```
====================== RESTART: D:\Python\ch6\ch6_32.py ======================
目前列表car内容 =  ['honda', 'bmw', 'toyota', 'ford']
使用sorted( )由大排到小
排序列表结果     =  ['toyota', 'honda', 'ford', 'bmw']
原先列表car内容 =  ['honda', 'bmw', 'toyota', 'ford']
目前列表num内容 =  [5, 3, 9, 2]
使用sorted( )由大排到小
排序列表结果     =  [9, 5, 3, 2]
原先列表num内容 =  [5, 3, 9, 2]
```

6-6 进阶列表操作

6-6-1 index( )

这个方法可以返回特定元素内容第一次出现的索引值，它的使用格式如下：

索引值 = 列表名称 .index(查找值)

如果查找值不存在，列表会出现错误。

程序实例 ch6_33.py：返回查找索引值的应用。

```
1  # ch6_33.py
2  cars = ['toyota', 'nissan', 'honda']
3  search_str = 'nissan'
4  i = cars.index(search_str)
5  print("查找元素 %s 第一次出现位置索引是 %d" % (search_str, i))
6  nums = [7, 12, 30, 12, 30, 9, 8]
7  search_val = 30
8  j = nums.index(search_val)
9  print("查找元素 %s 第一次出现位置索引是 %d" % (search_val, j))
```

执行结果

```
====================== RESTART: D:\Python\ch6\ch6_33.py ======================
查找元素 nissan 第一次出现位置索引是 1
查找元素 30 第一次出现位置索引是 2
```

如果查找值不在列表中会出现错误，所以在使用前建议可以先使用 in 表达式（可参考 6-10 节），先判断查找值是否在列表内，如果是在列表内，再执行 index( ) 方法。

程序实例 ch6_34.py：使用 ch6_16.py 的列表 James，这个列表有 Lebron James 的一系列比赛得分，由此列表计算他在第几场得最高分，同时列出所得分数。

```
1  # ch6_34.py
2  James = ['Lebron James',23, 19, 22, 31, 18] # 定义James列表
3  games = len(James)                          # 求元素数量
4  score_Max = max(James[1:games])             # 最高得分
5  i = James.index(score_Max)                  # 场次
6  print(James[0], "在第 %d 场得最高分 %d" % (i, score_Max))
```

执行结果

```
====================== RESTART: D:\Python\ch6\ch6_34.py ======================
Lebron James 在第 4 场得最高分 31
```

这个实例有一点不完美，因为如果有两场或更多场次得到相同分数的最高分，本程序无法处理，第 7 章将以实例讲解如何修改。

6-6-2 count( )

这个方法可以返回特定元素内容出现的次数，如果查找值不在列表会返回 0，它的使用格式如下：

次数 = 列表名称 .count(查找值)

程序实例 ch6_35.py：返回查找值出现的次数的应用。

```
1  # ch6_35.py
2  cars = ['toyota', 'nissan', 'honda']
3  search_str = 'nissan'
4  num1 = cars.count(search_str)
5  print("所查找元素 %s 出现 %d 次" % (search_str, num1))
6  nums = [7, 12, 30, 12, 30, 9, 8]
7  search_val = 30
8  num2 = nums.count(search_val)
9  print("所查找元素 %s 出现 %d 次" % (search_val, num2))
```

执行结果

```
====================== RESTART: D:\Python\ch6\ch6_35.py ======================
所查找元素 nissan 出现 1 次
所查找元素 30 出现 2 次
```

如果查找值不在列表会返回 0。

```
>>> x = [1,2,3]
>>> x.count(4)
0
```

6-7 列表内含列表

列表内含列表的基本形式如下：

num = [1, 2, 3, 4, 5, [6, 7, 8]]

对上述而言，num 是一个列表，在这个列表内有另一个列表 [7, 8, 9]，因为内部列表的索引值是 5，所以可以用 num[5] 获得这个元素列表的内容。

```
>>> num = [1, 2, 3, 4, 5, [6, 7, 8]]
>>> num[5]
[6, 7, 8]
>>>
```

如果想要存取列表内的列表元素，可以使用下列格式：

num[索引 1] [索引 2]

索引 1 是元素列表原先索引位置，索引 2 是元素列表内部的索引。

实例：列出列表内的列表元素值。

```
>>> num = [1, 2, 3, 4, 5, [6, 7, 8]]
>>> print(num[5][0])
6
>>> print(num[5][1])
7
>>> print(num[5][2])
8
>>>
```

列表内含列表的主要应用是，例如，可以用这个资料格式存储 NBA 球员 Lebron James 的数据如下所示：

```
James = [['Lebron James', 'SF','12/30/1984'], 23, 19, 22, 31, 18]
```

其中，第一个元素是列表，用于存储 Lebron James 的个人资料，其他则是存储每场得分数据。

程序实例 ch6_36.py：扩充 ch6_34.py，先列出 Lebron James 个人资料再计算哪一个场次得到最高分。程序第 2 行中 'SF' 全名是 Small Forward 即小前锋。

```
 1  # ch6_36.py
 2  James = [['Lebron James','SF','12/30/84'],23,19,22,31,18] # 定义James列表
 3  games = len(James)                                        # 求元素数量
 4  score_Max = max(James[1:games])                           # 最高得分
 5  i = James.index(score_Max)                                # 场次
 6  name = James[0][0]
 7  position = James[0][1]
 8  born = James[0][2]
 9  print("姓名      : ", name)
10  print("位置      : ", position)
11  print("出生日期 : ", born)
12  print("在第 %d 场得最高分 %d" % (i, score_Max))
```

执行结果

```
====================== RESTART: D:\Python\ch6\ch6_36.py ======================
姓名      :  Lebron James
位置      :  SF
出生日期 :  12/30/84
在第 4 场得最高分 31
```

程序实例 ch6_37.py：上述 ch6_36.py 的第 6 ～ 8 行是为了详细解说，真正了解 Python 精神的人，可以用下面一行取代这 3 行，用 Python 精神重新设计 ch6_36.py。

```
6  name, position, born = James[0]
```

执行结果 与 ch6_36.py 相同。

6-7-1　再谈 append( )

在 6-4-1 节提过了可以使用 append( ) 方法将元素插到列表的末端，其实也可以使用 append( ) 函数将某一列表插入到另一列表的末端，方法与插入元素方式相同，这时就会产生列表中有列表的效果。它的使用格式如下：

```
列表 A.append( 列表 B )                    # 列表 B 将接在列表 A 末端
```

程序实例 ch6_38.py：使用 append( ) 将列表插入另一列表的末端。

```
1  # ch6_38.py
2  cars1 = ['toyota', 'nissan', 'honda']
3  cars2 = ['ford', 'audi']
4  print("原先cars1列表内容 = ", cars1)
5  print("原先cars2列表内容 = ", cars2)
6  cars1.append(cars2)
7  print("执行append()后列表cars1内容 = ", cars1))
8  print("执行append()后列表cars2内容 = ", cars2))
```

执行结果

```
===================== RESTART: D:\Python\ch6\ch6_38.py =====================
原先cars1列表内容 =  ['toyota', 'nissan', 'honda']
原先cars2列表内容 =  ['ford', 'audi']
执行append( )后列表cars1内容 =  ['toyota', 'nissan', 'honda', ['ford', 'audi']]
执行append( )后列表cars2内容 =  ['ford', 'audi']
```

6-7-2 extend( )

这也是两个列表连接的方法，与 append( ) 类似，不过这个方法只适用两个列表连接，不能用于一般元素。同时在连接后，extend( ) 会将列表分解成元素，一一插入列表。它的使用格式如下：

列表 A.extend(列表 B)　　　　　　# 列表 B 将分解成元素插入列表 A 末端

程序实例 ch6_39.py：使用 extend( ) 方法取代 ch6_38.py，并观察执行结果。

```
1  # ch6_39.py
2  cars1 = ['toyota', 'nissan', 'honda']
3  cars2 = ['ford', 'audi']
4  print("原先cars1列表内容 = ", cars1)
5  print("原先cars2列表内容 = ", cars2)
6  cars1.extend(cars2)
7  print("执行extend()后列表cars1内容 = ", cars1)
8  print("执行extend()后列表cars2内容 = ", cars2)
```

执行结果

```
===================== RESTART: D:\Python\ch6\ch6_39.py =====================
原先cars1列表内容 =  ['toyota', 'nissan', 'honda']
原先cars2列表内容 =  ['ford', 'audi']
执行extend( )后列表cars1内容 =  ['toyota', 'nissan', 'honda', 'ford', 'audi']
执行extend( )后列表cars2内容 =  ['ford', 'audi']
```

上述语句执行后，cars1 将是含有 5 个元素的列表，每个元素都是字符串。

6-7-3 再看二维列表

所谓的二维列表（two dimension list）可以想成是二维空间，前面已有说明，本节将更进一步解说，下面是一个考试成绩系统的表格。

姓名	语文	英文	数学	总分
洪锦魁	80	95	88	0
洪冰儒	98	97	96	0
洪雨星	90	91	92	0
洪冰雨	91	93	95	0
洪星宇	92	97	90	0

上述总分先放 0，笔者会讲解如何处理这个部分，假设列表名称是 sc，在 Python 中可以用下列方式记录成绩系统。

```
sc = [['洪锦魁', 80, 95, 88, 0],
['洪冰儒', 98, 97, 96, 0],
['洪雨星', 90, 91, 92, 0],
['洪冰雨', 91, 93, 95, 0],
['洪星宇', 92, 97, 90, 0],
]
```

上述最后一个列表元素 ['洪星宇', 92, 97, 90, 0] 右边的“,”可有可无，这是 Python 设计人员贴心的设计，方便我们编辑这类应用，编译程序均可处理。

假设先不考虑表格的标题名称，设计程序时可以使用下列方式处理索引。

姓名	语文	英文	数学	总分
[0][0]	[0][1]	[0][2]	[0][3]	[0][4]
[1][0]	[1][1]	[1][2]	[1][3]	[1][4]
[2][0]	[2][1]	[2][2]	[2][3]	[2][4]
[3][0]	[3][1]	[3][2]	[3][3]	[3][4]
[4][0]	[4][1]	[4][2]	[4][3]	[4][4]

上述表格最常见的应用是，使用循环计算每个学生的总分，这将在第 7 章补充说明，在此将用现有的知识处理总分问题，为了简化只用两个学生姓名为实例说明。

程序实例 ch6_40.py：二维列表的成绩系统总分计算。

```
1  # ch6_40.py
2  sc = [['洪锦魁', 80, 95, 88, 0],
3        ['洪冰儒', 98, 97, 96, 0],
4       ]
5  sc[0][4] = sum(sc[0][1:4])
6  sc[1][4] = sum(sc[1][1:4])
7  print(sc[0])
8  print(sc[1])
```

执行结果

```
===================== RESTART: D:\Python\ch6\ch6_40.py =====================
['洪锦魁', 80, 95, 88, 263]
['洪冰儒', 98, 97, 96, 291]
```

6-8 列表的赋值与切片复制

6-8-1 列表赋值

假设我喜欢的运动是篮球与棒球，可以用下列方式设置列表：

```
mysports = ['basketball', 'baseball']
```

如果我的朋友也喜欢这两种运动，读者可能会想用下列方式设置列表。

```
friendsports = mysports
```

程序实例 ch6_41.py：列出我和朋友所喜欢的运动。

```
1  # ch6_41.py
2  mysports = ['basketball', 'baseball']
3  friendsports = mysports
4  print("我喜欢的运动      = ", mysports)
5  print("我朋友喜欢的运动 = ", friendsports)
```

执行结果

```
===================== RESTART: D:\Python\ch6\ch6_41.py =====================
我喜欢的运动      =  ['basketball', 'baseball']
我朋友喜欢的运动 =  ['basketball', 'baseball']
```

初看上述执行结果好像没有任何问题，可是如果我想加入美式足球 football 当作喜欢的运动，我的朋友想加入传统足球 soccer 当作喜欢的运动，这时我喜欢的运动如下：

basketball、baseball、football

我朋友喜欢的运动如下：

basketball、baseball、soccer

程序实例 ch6_42.py：继续使用 ch6_41.py，加入美式足球 football 当作喜欢的运动，我的朋友想加入传统足球 soccer 当作喜欢的运动，同时列出执行结果。

```
1  # ch6_42.py
2  mysports = ['basketball', 'baseball']
3  friendsports = mysports
4  print("我喜欢的运动      = ", mysports)
5  print("我朋友喜欢的运动 = ", friendsports)
6  mysports.append('football')
7  friendsports.append('soccer')
8  print("我喜欢的最新运动      = ", mysports)
9  print("我朋友喜欢的最新运动 = ", friendsports)
```

执行结果

```
===================== RESTART: D:\Python\ch6\ch6_42.py =====================
我喜欢的运动      =  ['basketball', 'baseball']
我朋友喜欢的运动 =  ['basketball', 'baseball']
我喜欢的最新运动      =  ['basketball', 'baseball', 'football', 'soccer']
我朋友喜欢的最新运动 =  ['basketball', 'baseball', 'football', 'soccer']
```

这时获得的结果，我和我的朋友喜欢的运动都相同，football 和 soccer 都变成两人共同喜欢的运动。类似这种只要有一个列表更改元素会影响到另一个列表同步更改，就是**赋值**的特性，所以使用时要小心。

6-8-2 地址的概念

在 2-2-2 节介绍了变量地址的意义，也可以应用于 Python 的其他数据类型，对于列表而言，如果使用下列方式设置两个列表变量相等，相当于只是将变量地址复制给另一个变量。

```
friendsports = mysports
```

上述语句相当于将 mysports 变量地址复制给 friendsport。所以程序实例 ch6_42.py 在执行时，两个列表变量所指的地址相同，在新增运动项目时，都是将运动项目加在同一变量地址，可参考下列实例。

程序实例 ch6_43.py：重新设计 ch6_42.py，增加列出列表变量的地址。

```
1  # ch6_43.py
2  mysports = ['basketball', 'baseball']
3  friendsports = mysports
4  print("列出mysports地址       = ", id(mysports))
5  print("列出friendsports地址 = ", id(friendsports))
6  print("我喜欢的运动       = ", mysports)
7  print("我朋友喜欢的运动 = ", friendsports)
8  mysports.append('football')
9  friendsports.append('soccer')
10 print(" -- 新增运动项目后 -- ")
11 print("列出mysports地址       = ", id(mysports))
12 print("列出friendsports地址 = ", id(friendsports))
13 print("我喜欢的最新运动       = ", mysports)
14 print("我朋友喜欢的最新运动 = ", friendsports)
```

执行结果

```
====================== RESTART: D:\Python\ch6\ch6_43.py ======================
列出mysports地址       =  18799272
列出friendsports地址 =  18799272
我喜欢的运动       =  ['basketball', 'baseball']
我朋友喜欢的运动 =  ['basketball', 'baseball']
 -- 新增运动项目后 --
列出mysports地址       =  18799272
列出friendsports地址 =  18799272
我喜欢的最新运动       =  ['basketball', 'baseball', 'football', 'soccer']
我朋友喜欢的最新运动 =  ['basketball', 'baseball', 'football', 'soccer']
```

由上述执行结果可以看到，使用程序第 3 行设置列表变量相等时，实际只是将列表地址复制给另一个列表变量。

6-8-3　列表的切片复制

切片复制（copy）的概念是，执行复制后**产生新列表对象**，当一个列表改变后，不会影响另一个列表的内容，这是本节的重点。方法如下：

```
friendsports = mysports[ : ]          # 切片复制
```

程序实例 ch6_44.py：使用切片复制方式，重新设计 ch6_42.py。下面是与 ch6_42.py 之间唯一不同的程序代码。

```
3  friendsports = mysports[:]
```

执行结果

```
====================== RESTART: D:\Python\ch6\ch6_44.py ======================
列出mysports地址       =  15915328
列出friendsports地址 =  15914888
我喜欢的运动       =  ['basketball', 'baseball']
我朋友喜欢的运动 =  ['basketball', 'baseball']
 -- 新增运动项目后 --
列出mysports地址       =  15915328
列出friendsports地址 =  15914888
我喜欢的最新运动       =  ['basketball', 'baseball', 'football']
我朋友喜欢的最新运动 =  ['basketball', 'baseball', 'soccer']
```

由上述执行结果可知，已经获得了两个列表彼此是不同的列表地址，同时也得到了想要的结果。

6-8-4　浅拷贝与深拷贝

在程序设计时，要复制另一个列表时，除了**赋值**（6-8-1）的概念，其实严格地说可以将**拷贝**分成**浅拷贝**（copy，有时也可以写成 shallow copy）与**深拷贝**（deepcopy）。

1. 赋值

假设 b=a，a 和 b 地址相同，指向对象彼此会联动，可以参考 6-8-1 节。

2. 浅拷贝

假设 b=a.copy( )，a 和 b 是独立的对象，但是它们的子对象元素是指向同一对象，也就是对象的子对象会联动。

实例 1：浅拷贝的应用，a 增加元素后观察结果。

```
>>> a = [1, 2, 3, [4, 5, 6]]
>>> b = a.copy()          ← 浅拷贝
>>> id(a), id(b)          ← 地址不同
(15518056, 49414872)
>>> a, b
([1, 2, 3, [4, 5, 6]], [1, 2, 3, [4, 5, 6]])
>>> a.append(7)           ← A增加元素
>>> a, b
([1, 2, 3, [4, 5, 6], 7], [1, 2, 3, [4, 5, 6]])
                      ↑
              a有更改, b没有更改
```

实例 2：浅拷贝的应用，a 的子对象增加元素后观察结果。

```
>>> a = [1, 2, 3, [4, 5, 6]]
>>> b = a.copy()
>>> a[3].append(7)
>>> a, b
([1, 2, 3, [4, 5, 6, 7]], [1, 2, 3, [4, 5, 6, 7]])
```

从上述执行结果可以发现 a 子对象因为指向同一地址，所以同时增加 7。

3. 深拷贝

假设 b=deepcopy(a)，a 和 b 以及其子对象都是独立的对象，所以未来不受干扰，使用前需要“import copy”模块，这是引用外部模块，后面会讲更多相关的应用。

实例 3：深拷贝的应用，并观察执行结果。

```
>>> import copy
>>> a = [1, 2, 3, [4, 5, 6]]
>>> b = copy.deepcopy(a)
>>> id(a), id(b)
(10293936, 15518496)
>>> a[3].append(7)
>>> a.append(8)
>>> a, b
([1, 2, 3, [4, 5, 6, 7], 8], [1, 2, 3, [4, 5, 6]])
```

由上述可以得到 b 完全不会受到 a 影响，深拷贝是得到完全独立的对象。

6-9 再谈字符串

3-4 节介绍了字符串（str）的概念，在 Python 的应用中可以将单一字符串当作一个序列，这个序列是由**字符**（character）所组成的，可想成**字符序列**。不过**字符串**与**列表**不同的是，字符串内的单一元素内容是不可更改的。

6-9-1　字符串的索引

可以使用**索引**值的方式取得**字符串**内容，索引方式与列表相同。

程序实例 ch6_45.py：使用正值与负值的索引列出字符串元素内容。

```
1  # ch6_45.py
2  string = "Python"
3  # 正值索引
4  print(" string[0] = ", string[0],
5        "\n string[1] = ", string[1],
6        "\n string[2] = ", string[2],
7        "\n string[3] = ", string[3],
8        "\n string[4] = ", string[4],
9        "\n string[5] = ", string[5])
10 # 负值索引
11 print(" string[-1] = ", string[-1],
12       "\n string[-2] = ", string[-2],
13       "\n string[-3] = ", string[-3],
14       "\n string[-4] = ", string[-4],
15       "\n string[-5] = ", string[-5],
16       "\n string[-6] = ", string[-6])
17 # 多重指定观念
18 s1, s2, s3, s4, s5, s6 = string
19 print("多重指定观念的输出测试 = ",s1,s2,s3,s4,s5,s6)
```

执行结果

```
====================== RESTART: D:\Python\ch6\ch6_45.py ======================
 string[0] =  P
 string[1] =  y
 string[2] =  t
 string[3] =  h
 string[4] =  o
 string[5] =  n
 string[-1] =  n
 string[-2] =  o
 string[-3] =  h
 string[-4] =  t
 string[-5] =  y
 string[-6] =  P
多重指定观念的输出测试 =  P y t h o n
```

6-9-2　字符串切片

6-1-3 节列表切片的概念可以应用于字符串，下面将直接以实例说明。

程序实例 ch6_46.py：字符串切片的应用。

```
1  # ch6_46.py
2  string = "Deep Learning"                    # 定义字符串
3  print("打印string第0-2元素      = ", string[0:3])
4  print("打印string第1-3元素      = ", string[1:4])
5  print("打印string第1,3,5元素    = ", string[1:6:2])
6  print("打印string第1到最后元素 = ", string[1:])
7  print("打印string前3元素        = ", string[0:3])
8  print("打印string后3元素        = ", string[-3:])
```

执行结果

```
===================== RESTART: D:\Python\ch6\ch6_46.py =====================
打印string第0-2元素      =  Dee
打印string第1-3元素      =  eep
打印string第1,3,5元素    =  epL
打印string第1到最后元素 =  eep Learning
打印string前3元素        =  Dee
打印string后3元素        =  ing
```

6-9-3 函数或方法

除了会变动内容的列表**函数或方法**不可应用于字符串外，其他则可以用于字符串。

函数	说明
len( )	计算字符串长度
max( )	最大值
min( )	最小值

程序实例 ch6_47.py：将函数 len( )、max( )、min( ) 应用于字符串。

```
1  # ch6_47.py
2  string = "Deep Learning"                    # 定义字符串
3  strlen = len(string)
4  print("字符串长度", strlen)
5  maxstr = max(string)
6  print("字符串最大的Unicode码值和字符", ord(maxstr), maxstr)
7  minstr = min(string)
8  print("字符串最小的Unicode码值和字符", ord(minstr), minstr)
```

执行结果

```
===================== RESTART: D:\Python\ch6\ch6_47.py =====================
字符串长度 13
字符串最大的Unicode码值和字符 114 r
字符串最小的Unicode码值和字符 32
```

6-9-4 将字符串转成列表

list( ) 函数可以将参数内的对象转成列表，下面是字符串转为列表的实例。

```
>>> x = list('Deep Stone')
>>> print(x)
['D', 'e', 'e', 'p', ' ', 'S', 't', 'o', 'n', 'e']
>>>
```

6-9-5　切片赋值的应用

字符串本身无法用切片方式更改内容，但是将字符串改为列表后，就可以使用切片更改列表内容了，下面是延续 6-9-4 节的实例。

```
>>> x[5:] = 'Mind'
>>> print(x)
['D', 'e', 'e', 'p', ' ', 'M', 'i', 'n', 'd']
>>>
```

6-9-6　使用 split( ) 分割字符串

这个方法可以将字符串以空格或其他符号作为分隔符，将字符串拆开，变成一个列表。

```
str1.split( )          # 以空格当作分隔符将字符串拆开成列表
str2.split(ch)         # 以 ch 字符当作分隔符将字符串拆开成列表
```

变成列表后可以使用 len( ) 获得此列表的元素个数，相当于可以计算字符串是由多少个英文字母组成，由于中文字之间没有空格，所以本节所述方法只**适用于纯英文文件**。如果我们将一篇文章或一本书读至一个字符串变量，可以使用这个方法获得这一篇文章或这一本书的字数。

程序实例 ch6_48.py：将两种不同类型的字符串转成列表，其中，str1 使用空格当作分隔符，str2 使用“\”当作分隔符（因为这是转义字符，所以使用 \\），同时这个程序会列出这两个列表的元素数量。

```
1  # ch6_48.py
2  str1 = "Silicon Stone Education"
3  str2 = "D:\Python\ch6"
4
5  sList1 = str1.split()                    # 字符串转成列表
6  sList2 = str2.split("\\")                # 字符串转成列表
7  print(str1, " 列表内容是 ", sList1)       # 打印列表
8  print(str1, " 列表字数是 ", len(sList1))  # 打印字数
9  print(str2, " 列表内容是 ", sList2)       # 打印列表
10 print(str2, " 列表字数是 ", len(sList2))  # 打印字数
```

执行结果

```
====================== RESTART: D:\Python\ch6\ch6_48.py ======================
Silicon Stone Education  列表内容是  ['Silicon', 'Stone', 'Education']
Silicon Stone Education  列表字数是  3
D:\Python\ch6  列表内容是  ['D:', 'Python', 'ch6']
D:\Python\ch6  列表字数是  3
```

6-9-7　列表元素的组合 join( )

在网络爬虫设计的程序应用中，可能会常常使用 join( ) 方法将所获得的路径与文件名组合，它的语法格式如下：

连接字符串 .join(列表)

基本上列表元素会用连接字符串组成一个字符串。

程序实例 ch6_49.py：将列表内容连接。

```
1  # ch6_49.py
2  path = ['D:','ch6','ch6_49.py']
3  connect = '\\'                        # 路径分隔字符
4  print(connect.join(path))
5  connect = '*'                         # 普通字符
6  print(connect.join(path))
```

执行结果

```
====================== RESTART: D:\Python\ch6\ch6_49.py ======================
D:\ch6\ch6_49.py
D:*ch6*ch6_49.py
```

6-9-8 字符串的其他方法

本节将讲解下列字符串方法，其中，startswith( ) 和 endswith( ) 如果是真则返回 True，如果是伪则返回 False。

startswith( )：可以列出字符串起始文字是否是特定子字符串。

endswith( )：可以列出字符串结束文字是否是特定子字符串。

replace(ch1,ch2)：将 ch1 字符串由另一字符串取代。

程序实例 ch6_50.py：列出字符串“CIA”是不是起始或结束字符串，以及出现次数。最后这个程序会将 Linda 字符串用 Lxx 字符串取代。

```
1  # ch6_50.py
2  msg = '''CIA Mark told CIA Linda that the secret USB had given to CIA Peter'''
3  print("字符串开头是CIA: ", msg.startswith("CIA"))
4  print("字符串结尾是CIA: ", msg.endswith("CIA"))
5  print("CIA出现的次数: ",msg.count("CIA"))
6  msg = msg.replace('Linda','Lxx')
7  print("新的msg内容 : ", msg)
```

执行结果

```
====================== RESTART: D:\Python\ch6\ch6_50.py ======================
字符串开头是CIA:  True
字符串结尾是CIA:  False
CIA出现的次数:  3
新的msg内容 :  CIA Mark told CIA Lxx that the secret USB had given to CIA Peter
```

当有一本小说时，可以由此计算各个人物出现次数，也可由此判断哪些人是主角哪些人是配角。

6-10 in 和 not in 表达式

in 和 not in 主要是用于判断一个对象是否属于另一个对象，对象可以是字符串（string）、列表（list）、元组（tuple）（第 8 章介绍）、字典（dict）（第 9 章介绍）。它的语法格式如下：

```
boolean_value = obj1 in obj2          # 对象 obj1 在对象 obj2 内会返回 True
boolean_value = obj1 not in obj2      # 对象 obj1 不在对象 obj2 内会返回 True
```

程序实例 ch6_51.py：请输入字符，这个程序会判断字符是否在字符串内。

```
1  # ch6_51.py
2  password = 'deepstone'
3  ch = input("请输入字符 = ")
4  print("in表达式")
5  if ch in password:
6      print("输入字符在密码中")
7  else:
8      print("输入字符不在密码中")
9
10 print("not in表达式")
11 if ch not in password:
12     print("输入字符不在密码中")
13 else:
14     print("输入字符在密码中")
```

执行结果

```
====================== RESTART: D:\Python\ch6\ch6_51.py ======================
请输入字符 = d
in表达式
输入字符在密码中
not in表达式
输入字符在密码中
```

其实这个功能一般更常见是用在检测某个元素是否存在列表中，如果不存在，则将它加入列表内，可参考下列实例。

程序实例 ch6_52.py：这个程序基本上会要求输入一个水果名称，如果列表内目前没有这个水果，就将输入的水果加入列表内。

```
1  # ch6_52.py
2  fruits = ['apple', 'banana', 'watermelon']
3  fruit = input("请输入水果 = ")
4  if fruit in fruits:
5      print("这个水果已经有了")
6  else:
7      fruits.append(fruit)
8      print("谢谢提醒已经加入水果清单: ", fruits)
```

执行结果

```
====================== RESTART: D:\Python\ch6\ch6_52.py ======================
请输入水果 = orange
谢谢提醒已经加入水果清单:  ['apple', 'banana', 'watermelon', 'orange']
```

6-11 is 和 is not 表达式

可以用于比较两个对象是否相同，在此所谓相同并不只是内容相同，而是指对象变量指向相同的内存，对象可以是变量、字符串、列表、元组（第 8 章介绍）、字典（第 9 章介绍）。它的语法格式如下：

```
boolean_value = obj1 is obj2          # 对象 obj1 等于对象 obj2 内会返回 True
boolean_value = obj1 is not obj2      # 对象 obj1 不等于对象 obj2 内会返回 True
```

6-11-1　整数变量在内存地址的观察

在 2-2-2 节已经简单说明 id( ) 可以获得变量的地址，在 6-8-2 节已经讲解可以使用 id( ) 函数获得列表变量地址，其实这个函数也可以获得整数（或浮点数）变量在内存中的地址，当我们在 Python 程序中设立变量时，如果两个整数（或浮点数）变量内容相同，它们会使用相同的内存地址存储此变量。

程序实例 ch6_53.py：整数变量在内存地址的观察。这个程序比较特别的是，程序执行之初，变量 x 和 y 的值是 10，所以可以看到经过 id( ) 函数后，彼此有相同的内存位置。变量 z 和 r 由于值与 x 和 y 不相同，所以有不同的内存地址，经过第 9 行运算后，r 的值变为 10，最后得到 x、y 和 r 不仅值相同同时也指向相同的内存地址。

```
1  # ch6_53.py
2  x = 10
3  y = 10
4  z = 15
5  r = 20
6  print("x = %d, y = %d, z = %d, r = %d" % (x, y, z, r))
7  print("x地址 = %d, y地址 = %d, z地址 = %d, r地址 = %d"
8        % (id(x), id(y), id(z), id(r)))
9  r = x                                   # r的值将变为10
10 print("x = %d, y = %d, z = %d, r = %d" % (x, y, z, r))
11 print("x地址 = %d, y地址 = %d, z地址 = %d, r地址 = %d"
12       % (id(x), id(y), id(z), id(r)))
```

执行结果

```
====================== RESTART: D:\Python\ch6\ch6_53.py ======================
x = 10, y = 10, z = 15, r = 20
x地址 = 1349175568, y地址 = 1349175568, z地址 = 1349175648, r地址 = 1349175728
x = 10, y = 10, z = 15, r = 10
x地址 = 1349175568, y地址 = 1349175568, z地址 = 1349175648, r地址 = 1349175568
```

当 r 变量值变为 10 时，它所指的内存地址与 x 和 y 变量相同了。

6-11-2　将 is 和 is not 表达式应用于整数变量

程序实例 ch6_54.py：is 和 is not 表达式应用于整数变量。

```
1  # ch6_54.py
2  x = 10
3  y = 10
4  z = 15
5  r = z - 5
6  boolean_value = x is y
7  print("x位址 = %d, y位址 = %d" % (id(x), id(y)))
8  print("x = %d, y = %d, " % (x, y), boolean_value)
9
10 boolean_value = x is z
11 print("x位址 = %d, z位址 = %d" % (id(x), id(z)))
12 print("x = %d, z = %d, " % (x, z), boolean_value)
13
14 boolean_value = x is r
15 print("x位址 = %d, r位址 = %d" % (id(x), id(r)))
16 print("x = %d, r = %d, " % (x, r), boolean_value)
17
18 boolean_value = x is not y
19 print("x位址 = %d, y位址 = %d" % (id(x), id(y)))
20 print("x = %d, y = %d, " % (x, y), boolean_value)
```

```
21
22  boolean_value = x is not z
23  print("x位址 = %d, z位址 = %d" % (id(x), id(z)))
24  print("x = %d, z = %d, " % (x, z), boolean_value)
25
26  boolean_value = x is not r
27  print("x位址 = %d, r位址 = %d" % (id(x), id(r)))
28  print("x = %d, r = %d, " % (x, r), boolean_value)
```

执行结果

```
====================== RESTART: D:/Python/ch6/ch6_54.py ======================
x位址 = 1668626832, y位址 = 1668626832
x = 10, y = 10,  True
x位址 = 1668626832, z位址 = 1668626912
x = 10, z = 15,  False
x位址 = 1668626832, r位址 = 1668626832
x = 10, r = 10,  True
x位址 = 1668626832, y位址 = 1668626832
x = 10, y = 10,  False
x位址 = 1668626832, z位址 = 1668626912
x = 10, z = 15,  True
x位址 = 1668626832, r位址 = 1668626832
x = 10, r = 10,  False
```

6-11-3　将 is 和 is not 表达式应用于列表变量

程序实例 ch6_55.py：这个范例所使用的 3 个列表内容均相同，但是 mysports 和 sports1 所指地址相同所以会被视为相同对象，sports2 则指向不同地址所以会被视为不同对象，在使用 is 指令测试时，不同地址的列表会被视为不同的列表。

```
1   # ch6_55.py
2   mysports = ['basketball', 'baseball']
3   sports1 = mysports            # 赋值
4   sports2 = mysports[:]         # 切片复制新列表
5   print("我喜欢的运动 = ", mysports, "地址是 = ", id(mysports))
6   print("运动 1       = ", sports1,  "地址是 = ", id(sports1))
7   print("运动 2       = ", sports2,  "地址是 = ", id(sports2))
8   boolean_value = mysports is sports1
9   print("我喜欢的运动 is 运动 1     = ", boolean_value)
10
11  boolean_value = mysports is sports2
12  print("我喜欢的运动 is 运动 2     = ", boolean_value)
13
14  boolean_value = mysports is not sports1
15  print("我喜欢的运动 is not 运动 1 = ", boolean_value)
16
17  boolean_value = mysports is not sports2
18  print("我喜欢的运动 is not 运动 2 = ", boolean_value)
```

执行结果

```
====================== RESTART: D:\Python\ch6\ch6_55.py ======================
我喜欢的运动 =  ['basketball', 'baseball'] 地址是 =  43506304
运动 1       =  ['basketball', 'baseball'] 地址是 =  43506304
运动 2       =  ['basketball', 'baseball'] 地址是 =  43505664
我喜欢的运动 is 运动 1     =  True
我喜欢的运动 is 运动 2     =  False
我喜欢的运动 is not 运动 1 =  False
我喜欢的运动 is not 运动 2 =  True
```

6-11-4　将 is 应用于 None

在 5-7 节介绍了 None，None 是一个尚未定义的值，这是 NoneType 数据类型，在布尔值中会

被视为 False，但是并不是空值，可以用下列实例做测试。

实例：测试 None 并不是空的。

```
>>> x = []
>>> if x is None:
        print("It is None")
else:
        print("It is not None")


It is not None
```

上述概念可以应用于 Python 其他数据结构上，如元组、字典、集合等。

6-12 enumerate 对象

enumerate( ) 方法可以将 iterable（迭代）类数值的**元素**用**索引值**与**元素**配对方式返回，返回的数据称为 **enumerate 对象**，特别是用这个方式可以为可迭代对象的每个元素增加索引值，这对未来的数据应用是有帮助的。其中，iterable 类数值可以是**列表**、**元组**（第 8 章说明）、**集合**（set）（第 10 章说明）等。它的语法格式如下：

```
obj = enumerate(iterable[, start = 0])      # 若省略 start = 设置，默认索引
                                               值是 0
```

注：第 7 章介绍完循环的概念，会针对**可迭代对象**（iterable object）做更进一步说明。

未来我们可以使用 list( ) 将 enumerate 对象转成**列表**，使用 tuple( ) 将 enumerate 对象转成**元组**（第 8 章说明）。

程序实例 ch6_56.py：将列表数据转成 enumerate 对象，同时列出此对象类型。

```
1  # ch6_56.py
2  drinks = ["coffee", "tea", "wine"]
3  enumerate_drinks = enumerate(drinks)          # 数值初始是0
4  print(enumerate_drinks)                       # 返回enumerate对象所在内存
5  print("下列是输出enumerate对象类型")
6  print(type(enumerate_drinks))                 # 列出对象类型
```

执行结果

```
===================== RESTART: D:\Python\ch6\ch6_56.py =====================
<enumerate object at 0x02F0EB48>
下列是输出enumerate对象类型
<class 'enumerate'>
```

程序实例 ch6_57.py：将列表数据转成 enumerate 对象，再将 enumerate 对象转成列表的实例，start 索引起始值分别为 0 和 10。

```
1  # ch6_57.py
2  drinks = ["coffee", "tea", "wine"]
3  enumerate_drinks = enumerate(drinks)                  # 数值初始是0
4  print("转成列表输出, 初始索引值是 0 = ", list(enumerate_drinks))
5
6  enumerate_drinks = enumerate(drinks, start = 10)     # 数值初始是10
7  print("转成列表输出, 初始索引值是10 = ", list(enumerate_drinks))
```

执行结果

```
====================== RESTART: D:\Python\ch6\ch6_57.py ======================
转成列表输出，初始索引值是 0 =  [(0, 'coffee'), (1, 'tea'), (2, 'wine')]
转成列表输出，初始索引值是10 =  [(10, 'coffee'), (11, 'tea'), (12, 'wine')]
```

上述程序第 4 行的 list( ) 函数可以将 enumerate 对象转成列表，从打印的结果可以看到每个列表对象元素已经增加索引值了。在第 7 章介绍完循环后，7-5 节还将继续使用循环解析 enumerate 对象。

6-13　专题——建立大型列表 / 用户账号管理系统 / 文件加密

6-13-1　制作大型的列表数据

有时我们想要制作更大型的列表数据结构，例如，列表的元素是列表，可以参考下列实例。

实例：列表的元素是列表。

```
>>> asia = ['Beijing', 'Hongkong', 'Tokyo']
>>> usa = ['Chicago', 'New York', 'Hawaii', 'Los Angeles']
>>> europe = ['Paris', 'London', 'Zurich']
>>> world = [asia, usa, europe]
>>> type(world)
<class 'list'>
>>> world
[['Beijing', 'Hongkong', 'Tokyo'], ['Chicago', 'New York', 'Hawaii', 'Los Angele
s'], ['Paris', 'London', 'Zurich']]
```

6-13-2　用户账号管理系统

一个公司或学校的计算机系统，一定有一个账号管理系统，要进入系统需要登录账号，如果你是这个单位设计账号管理系统的人，可以将账号存储在列表内。然后可以使用 in 功能判断用户输入账号是否正确。

程序实例 ch6_58.py：设计一个账号管理系统，这个程序分成两个部分，第一个部分是建立账号，读者的输入将会存在 accounts 列表中。第二个部分是要求输入账号，如果输入正确会输出“欢迎进入深石系统”，如果输入错误会输出“账号错误”。

```
1  # ch6_58.py
2  accounts = []                           # 建立空账号列表
3  account = input("请输入新账号 = ")
4  accounts.append(account)                # 将输入加入账号列表
5
6  print("深石公司系统")
7  ac = input("请输入账号 = ")
8  if ac in accounts:
9      print("欢迎进入深石系统")
10 else:
11     print("账号错误")
```

执行结果

```
===================== RESTART: D:\Python\ch6\ch6_58.py =====================
请输入新账号 = deep
深石公司系统
请输入账号 = deep
欢迎进入深石系统
>>>
===================== RESTART: D:\Python\ch6\ch6_58.py =====================
请输入新账号 = deep
深石公司系统
请输入账号 = kwei
账号错误
```

6-13-3 文件加密

这一节将简单介绍切片的奥妙，然后讲解文件加密的精神，未来当读者学会更多 Python 知识时，还会扩充至实际设计一个加密程序。

其实最简单的加密是将每个英文字母往前移，对应至不同字母，只要记住所对应的字母，就可以解密。例如，将每个英文字母往前移 3 个次序，实例是将 D 对应 A，E 对应 B，F 对应 C，原先的 A 对应 X，B 对应 Y，C 对应 Z。整个思路如下所示。

D	E	F	G	…	Y	Z	A	B	C
↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
A	B	C	D	…	V	W	X	Y	Z

所以现在需要的就是设计“DEF … ABC”字母可以对应“ABC … XYZ”，可以参考下列实例完成。或是让“ ABC … XYZ”对应“ DEF … ABC”也可以。

实例：建立 ABC … Z 字母的字符串，然后使用切片取得前 3 个英文字母与后 23 个英文字母，最后组合，可以得到新的字母排序。

```
>>> abc = 'ABCDEFGHIJKLMNOPQRSTUVWYZ'
>>> front3 = abc[:3]
>>> end23 = abc[3:]
>>> subText = end23 + front3
>>> print(subText)
DEFGHIJKLMNOPQRSTUVWYZABC
```

在第 9 章还会扩充此概念。

习题

1. 考试成绩分数分别是 87,99,69,52,78,98,80,92，请列出最高分、最低分、总分、平均分。（6-1 节）

```
===================== RESTART: D:/Python/ex/ex6_1.py =====================
最高分 =  99
最低分 =  52
总分   =  655
平均   =  81.88
```

2. 一家汽车经销商原本可以销售 Toyota、Nissan、Honda，现在 Nissan 销售权被回收，改成销售 Ford，可用下列方式设计销售品牌。（6-1 节）

```
===================== RESTART: D:/Python/ex/ex6_2.py =====================
旧汽车销售品牌 ['Toyota', 'Nissan', 'Honda']
新汽车销售品牌 ['Toyota', 'Ford', 'Honda']
```

3. 有 str1、str2、str3 字符串内容如下：(6-2 节)

```
str1 = '  Python  '
str2 = 'is  '
str3 = '  easy'
```

请使用 strip( )、rstrip( )、lstrip( ) 处理成下列输出。

```
'Python is easy'
```

```
====================== RESTART: D:/Python/ex/ex6_3.py ======================
Python is easy
```

4 . 请建立 5 个城市，然后分别执行下列工作。(6-4 节)

（1）列出这 5 个城市。

（2）请在最后位置增加 London。

（3）请在中央位置增加 Xian。

（4）请使用 remove( ) 方法删除 Tokyo。

```
====================== RESTART: D:/Python/ex/ex6_4.py ======================
['Taipei', 'Beijing', 'Tokyo', 'Chicago', 'Nanjing']
['Taipei', 'Beijing', 'Tokyo', 'Chicago', 'Nanjing', 'London']
['Taipei', 'Beijing', 'Tokyo', 'Xian', 'Chicago', 'Nanjing', 'London']
['Taipei', 'Beijing', 'Xian', 'Chicago', 'Nanjing', 'London']
```

5. 请在屏幕输入 5 个考试成绩，然后执行下列工作。(6-5 节)

（1）列出分数列表。

（2）高分往低分排列。

（3）低分往高分排列。

（4）列出最高分。

（5）列出总分。

```
====================== RESTART: D:/Python/ex/ex6_5.py ======================
请输入5个考试成绩 : 87, 90, 76, 85, 92
分数列表          :  [87, 90, 76, 85, 92]
高分往低分排列 :  [92, 90, 87, 85, 76]
高分往低分排列 :  [76, 85, 87, 90, 92]
最高分            :  92
总分              :  430
```

6. 请参考 6-7-3 节内容的数据与 ch6_40.py，将学生增加为 5 人，同时增加平均分字段，平均分数取到小数点第 1 位。(6-7 节)

```
====================== RESTART: D:/Python/ex/ex6_6.py ======================
['洪锦魁', 80, 95, 88, 263, 87.7]
['洪冰儒', 98, 97, 96, 291, 97.0]
['洪雨星', 91, 93, 95, 279, 93.0]
['洪冰雨', 92, 94, 90, 276, 92.0]
['洪星宇', 92, 97, 90, 279, 93.0]
```

7. 有一个字符串如下：(6-9 节)

```
FBI Mark told CIA Linda that the secret USB had given to FBI Peter
```

（1）请列出 FBI 出现的次数。

（2）请将 FBI 字符串用 XX 取代。

```
====================== RESTART: D:/Python/ex/ex6_7.py ======================
FBI出现的次数:  2
新的msg内容 :  XX Mark told CIA Linda that the secret USB had given to XX Peter
```

8. 输入一个字符串，这个程序可以判断这是否是网址字符串。(6-9 节)

提示：网址字符串是以“http://”或“https://”字符串开头。

```
======================= RESTART: D:/Python/ex/ex6_8.py =======================
请输入网址 : http://www.siliconstone.com
网址格式正确
>>>
======================= RESTART: D:/Python/ex/ex6_8.py =======================
请输入网址 : ILoveDeepMind
网址格式错误
```

9. 有一首法国儿歌，也是我们小时候唱的《两只老虎》，歌曲内容如下。(6-9 节)

Are you sleeping, are you sleeping, Brother John, Brother John?
Morning bells are ringing, morning bells are ringing.
Ding ding dong, Ding ding dong.

请在建立上述字符串时省略标点符号，最后列出此字符串。然后将字符串转为列表，同时列出列表，首先列出**歌曲的字数**，然后在屏幕上输入字符串，程序可以列出这个**字符串出现次数**。

```
======================= RESTART: D:/Python/ex/ex6_9.py =======================
歌曲字符串内容
Are you sleeping are you sleeping Brother John Brother John
Morning bells are ringing morning bells are ringing
Ding ding dong Ding ding dong
歌曲列表内容
['are', 'you', 'sleeping', 'are', 'you', 'sleeping', 'brother', 'john', 'brother
', 'john', 'morning', 'bells', 'are', 'ringing', 'morning', 'bells', 'are', 'rin
ging', 'ding', 'ding', 'dong', 'ding', 'ding', 'dong']
歌曲的字数 : 24
请输入字符串 : ding
ding 出现的 4 次
```

10. 本书 1-11 节有 Python 之禅的内容，请将该内容当作字符串，然后将该内容以行为单位当作列表元素，先列出 Python 之禅的内容，然后列出列表内容。

```
====================== RESTART: D:/Python/ex/ex6_10.py ======================
下列是Python之禅内容
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
以换行符将Python 之禅的行数据变成列表元素
['The Zen of Python, by Tim Peters', '', 'Beautiful is better than ugly.', 'Expl
icit is better than implicit.', 'Simple is better than complex.', 'Complex is be
tter than complicated.', 'Flat is better than nested.', 'Sparse is better than d
ense.', 'Readability counts.', "Special cases aren't special enough to break the
 rules.", 'Although practicality beats purity.', 'Errors should never pass silen
tly.', 'Unless explicitly silenced.', 'In the face of ambiguity, refuse the temp
tation to guess.', 'There should be one-- and preferably only one --obvious way
to do it.', "Although that way may not be obvious at first unless you're Dutch."
, 'Now is better than never.', 'Although never is often better than *right* now.
', "If the implementation is hard to explain, it's a bad idea.", 'If the impleme
ntation is easy to explain, it may be a good idea.', "Namespaces are one honking
 great idea -- let's do more of those!"]
```

11. 请建立一个晚会宴客名单，有 3 个数据“Mary、Josh、Tracy”。请做一个选单，每次执行都会列出目前邀请名单，同时有选单，如果选择 1，可以增加一位邀请名单；如果选择 2，可以删除一

位邀请名单。以目前所学命令，执行程序一次只能调整一次，如果删除名单时输入错误，则列出**名单输入错误**。(6-10 节)

```
===================== RESTART: D:/Python/ex/ex6_11.py =====================
目前宴会名单 ['Mary', 'Josh', 'Tracy']
1:增加名单
2:删除名单
 = 1
请输入名字 : Kevin
新的宴会名单 :  ['Mary', 'Josh', 'Tracy', 'Kevin']
>>>
===================== RESTART: D:/Python/ex/ex6_11.py =====================
目前宴会名单 ['Mary', 'Josh', 'Tracy']
1:增加名单
2:删除名单
 = 2
请输入名字 : Mary
新的宴会名单 :  ['Josh', 'Tracy']
>>>
===================== RESTART: D:/Python/ex/ex6_11.py =====================
目前宴会名单 ['Mary', 'Josh', 'Tracy']
1:增加名单
2:删除名单
 = 2
请输入名字 : Tom
名单输入错误
```

12. 请修改 6-13-2 节的加密实例，字符串 abc 改为“abc … z”，修改方式如下。(6-13 节)

f	g	h	i	…	a	b	c	d	e
↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
a	b	c	d	…	v	w	x	y	z

最后打印出 abc 与 subText。

```
===================== RESTART: D:/Python/ex/ex6_13.py =====================
abcdefghijklmnopqrstuvwxyz
fghijklmnopqrstuvwxyzabcde
```

07

第 7 章

循环设计

本章摘要

7-1 基本 for 循环
7-2 range( ) 函数
7-3 进阶的 for 循环应用
7-4 while 循环
7-5 enumerate 对象使用 for 循环解析
7-6 专题——购物车设计 / 成绩系统 / 圆周率

假设笔者要求读者设计一个从 1 加到 10 的程序，然后打印结果，读者可能用下列方式设计这个程序。

程序实例 ch7_1.py：从 1 加到 10，同时打印结果。

```
1  # ch7_1.py
2  sum = 1+2+3+4+5+6+7+8+9+10
3  print("总和 = ", sum)
```

执行结果

```
===================== RESTART: D:\Python\ch7\ch7_1.py =====================
总和 =  55
```

如果笔者要求读者从 1 加到 100 或 1000，此时，若是仍用上述方法设计程序，就显得很不经济。

另一种状况，如果一个数据库列表内含 1000 个客户的名字，现在要举办晚宴，所以要打印客户姓名，如果用下列方式设计，将是很不实际的行为。

程序实例 ch7_2.py：一个不完整且不切实际的程序。

```
1  # ch7_2.py -- 不完整的程序
2  vipNames = ['James','Linda','Peter', ... , 'Kevin']
3  print("客户1 = ", vipNames[0])
4  print("客户2 = ", vipNames[1])
5  print("客户3 = ", vipNames[2])
6  ...
7  ...
8  print("客户999 = ", vipNames[999])
```

你的程序可能要写超过 1000 行。当然，碰上这类问题时，是不可能用上述方法处理的，Python 语言提供了解决这类问题的方式，即**循环**，这也是本章的主题。

7-1 基本 for 循环

for 循环可以让程序将整个**对象**内的元素**遍历**（也称**迭代**），在遍历期间，同时可以记录或输出每次遍历的状态（或称**轨迹**）。例如，第 2 章的计算银行复利问题，在该章节由于尚未介绍循环的概念，无法记录每一年的本金和，有了本章的概念就可以轻松记录每一年的本金和变化。for 循环基本语法格式如下：

```
for var in 可迭代对象 :            # 可迭代对象英文是 iterable object
    程序代码区块
```

可迭代对象（iterable object）可以是**列表**、**元组**、**字典**与**集合**或 range( )，在信息科学中**迭代**（iteration）可以解释为**重复执行语句**。上述语法可以解释为将可迭代对象的元素当作 var，重复执行，直到每个元素都被执行一次，整个循环才会停止。

设计上述程序代码区块时，必须要留意缩排的问题，可以参考 if 语句格式。由于目前笔者只介绍列表，所以读者可以想象这个**可迭代对象**（iterable）是**列表**，第 8 章会讲解**元组**，第 9 章会讲解**字典**，第 10 章会讲解**集合**。另外，上述 for 循环的**可迭代对象**也常是 range( ) 函数产生的**可迭代对**

象，将在 7-2 节说明。

7-1-1 for 循环基本操作

例如，如果一个 NBA 球队有 5 位球员，分别是 Curry、Jordan、James、Durant、Obama，现在想列出这 5 位球员，那么就很适合使用 for 循环执行这个工作。

程序实例 ch7_3.py：列出球员名称。

```
1  # ch7_3.py
2  players = ['Curry', 'Jordan', 'James', 'Durant', 'Obama']
3  for player in players:
4      print(player)
```

执行结果

```
====================== RESTART: D:/Python/ch7/ch7_3.py ======================
Curry
Jordan
James
Durant
Obama
```

上述程序执行的过程是，当第一次执行下列语句时：

```
for player in players:
```

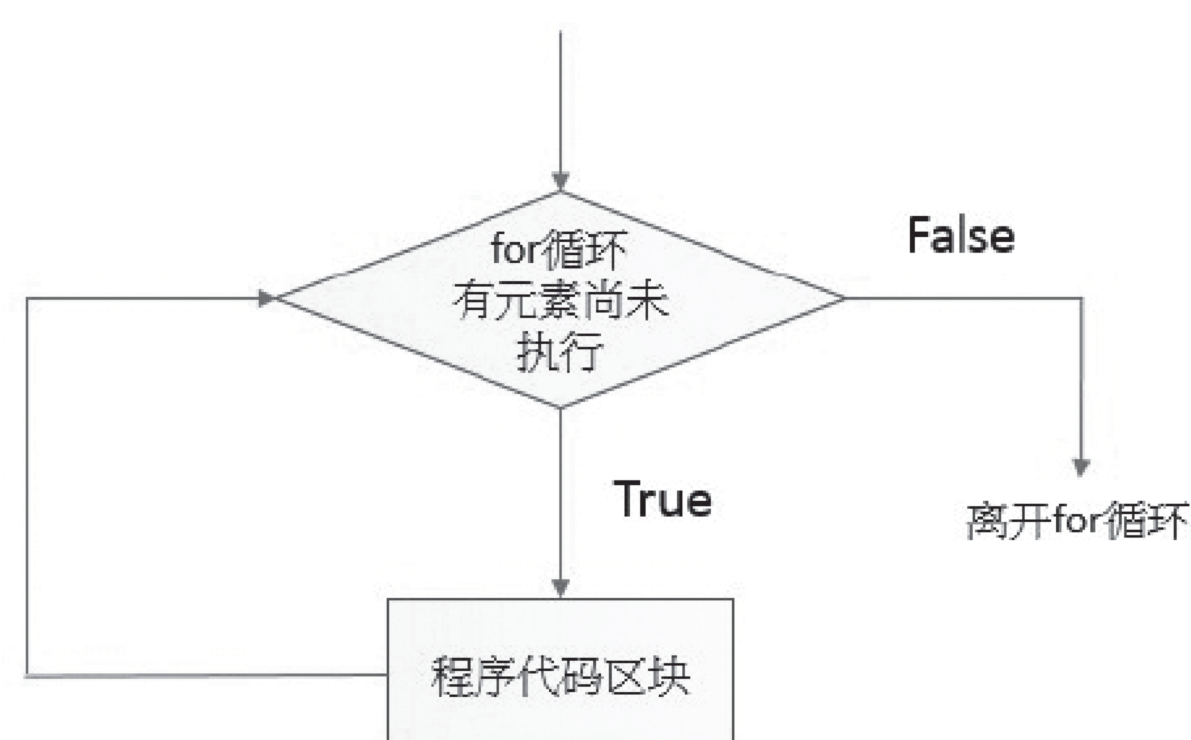
player 的内容是‘Curry’，然后执行 print(player)，所以会打印‘Curry’，也可以将此过程称为**第一次迭代**。由于列表 players 内还有其他元素尚未执行，所以会执行**第二次迭代**，当执行**第二次迭代**到下列语句时：

```
for player in players:
```

player 的内容是‘Jordan’，然后执行 print(player)，所以会打印‘Jordan’。由于列表 players 内还有其他元素尚未执行，所以会执行**第三次迭代**，……，当执行**第五次迭代**到下列语句时：

```
for player in players:
```

player 的内容是‘Obama’，然后执行 print(player)，所以会打印‘Obama’。第六次要执行 for 循环时，由于列表 players 内所有元素已经执行，所以这个循环就算执行结束了。下面是循环的流程示意图。



7-1-2 程序代码区块只有一行

使用 for 循环时，如果程序代码区块只有一行，它的语法格式如下：

for var in 可迭代对象：程序代码区块

程序实例 ch7_4.py：重新设计 ch7_3.py。

```
1  # ch7_4.py
2  players = ['Curry', 'Jordan', 'James', 'Durant', 'Obama']
3  for player in players: print(player)
```

执行结果 与 ch7_3.py 相同。

7-1-3　程序代码区块有多行

如果 for 循环的程序代码区块有多行语句时，要留意这些语句同时需要做缩排处理。它的语法格式如下：

for var in 可迭代对象：
程序代码
程序代码
…

程序实例 ch7_5.py：这个程序在设计时，首先将列表的元素英文名字全部改成小写，然后 for 循环的程序代码区块是有两行，这两行（第 4 和 5 行）都需内缩处理，player.title( ) 的 title( ) 方法可以处理第一个字母以大写显示。

```
1  # ch7_5.py
2  players = ['curry', 'jordan', 'james', 'durant', 'obama']
3  for player in players:
4      print(player.title( ) + ", it was a great game.")
5      print("我迫不及待想看下一场比赛, " + player.title( ))
```

执行结果

```
==================== RESTART: D:\Python\ch7\ch7_5.py ====================
Curry, it was a great game.
我迫不及待想看下一场比赛, Curry
Jordan, it was a great game.
我迫不及待想看下一场比赛, Jordan
James, it was a great game.
我迫不及待想看下一场比赛, James
Durant, it was a great game.
我迫不及待想看下一场比赛, Durant
Obama, it was a great game.
我迫不及待想看下一场比赛, Obama
```

7-1-4　将 for 循环应用于列表区间元素

Python 也允许将 for 循环应用于 6-1-2 节和 6-1-3 节所截取的区间列表元素上。

程序实例 ch7_6.py：列出列表前 3 位和后 3 位的球员名称。

```
1  # ch7_6.py
2  players = ['Curry', 'Jordan', 'James', 'Durant', 'Obama']
3  print("打印前3位球员")
4  for player in players[:3]:
5      print(player)
6  print("打印后3位球员")
7  for player in players[-3:]:
8      print(player)
```

执行结果

```
======================= RESTART: D:\Python\ch7\ch7_6.py =======================
打印前3位球员
Curry
Jordan
James
打印后3位球员
James
Durant
Obama
```

这个概念其实很有用，例如，如果你设计一个学习网站，想要每天列出前 3 名学生的基本数据同时表扬，可以将每个人的学习成果放在列表内，同时用**降序**排序方式处理，最后可用本节概念列出前 3 名学生的资料。

注 **升序**是指由小到大排列，**降序**是指由大到小排列。

7-1-5 将 for 循环应用于数据类别的判断

程序实例 ch7_7.py：有一个 files 列表内含一系列文件名，请将“.py”的 Python 程序文件另外建立到 py 列表，然后打印。

```
1  # ch7_7.py
2  files = ['da1.c','da2.py','da3.py','da4.java']
3  py = []
4  for file in files:
5      if file.endswith('.py'):     # 以.py为扩展名
6          py.append(file)          # 加入列表
7  print(py)
```

执行结果

```
======================= RESTART: D:/Python/ch7/ch7_7.py =======================
['da2.py', 'da3.py']
```

程序实例 ch7_8.py：有一个列表 names，元素内容是姓名，请将姓洪的成员建立在 lastname 列表内，然后打印。

```
1  # ch7_8.py
2  names = ['洪锦魁','洪冰儒','东霞','大成']
3  lastname = []
4  for name in names:
5      if name.startswith('洪'):    # 是否姓氏洪开头
6          lastname.append(name)    # 加入列表
7  print(lastname)
```

执行结果

```
======================= RESTART: D:\Python\ch7\ch7_8.py =======================
['洪锦魁', '洪冰儒']
```

7-1-6 删除列表内重复的元素

程序实例 ch7_9.py：删除列表 fruits2 内和 fruits1 内已有的元素，我们可以使用 for 循环完成此

工作。

```
1  # ch7_9.py
2  fruits1 = ['苹果', '香蕉', '西瓜', '水蜜桃', '百香果']
3  fruits2 = ['香蕉', '番石榴', '西瓜']
4  print("目前fruits2列表 : ", fruits2)
5  for fruit in fruits2[:]:
6      if fruit in fruits1:
7          fruits2.remove(fruit)
8          print("删除 %s " % fruit)
9  print("最后fruits2列表 : ", fruits2)
```

执行结果

```
====================== RESTART: D:\Python\ch7\ch7_9.py ======================
目前fruits2列表 :  ['香蕉', '番石榴', '西瓜']
删除 香蕉
删除 西瓜
最后fruits2列表 :  ['番石榴']
>>>
```

7-1-7　活用 for 循环

在 6-2-4 节实例 1 中列出了字符串的相关方法，其实也可以使用 for 循环一一列出它们。

实例：列出字符串的方法，下面只列出部分方法。

```
>>> string = 'abc'
>>> for i in dir(string):
        print(i)


__add__
__class__
__contains__
```

7-2　range( ) 函数

Python 可以使用 range( ) 函数产生一个等差数列，我们又称这个等差数列为**可迭代对象**（iterable object），也可以称为 range 对象。由于 range( ) 是产生等差数列，可以直接使用，将此等差数列当作循环的计数器。

在 7-1 节使用“for var in **可迭代对象**”当作循环，这时会使用**可迭代对象元素**当作循环指针，如果是要迭代对象内的元素，这是好方法。但是如果只是要执行普通的循环迭代，由于可迭代对象占用一些内存空间，所以这类循环需要用较多系统资源。这时应该直接使用 range( ) 对象，这类迭代只有迭代时的计数指针需要内存，所以可以省略内存空间。range( ) 的用法与列表的切片（slice）类似。

range(**start**, stop, **step**)

上述语句中 stop 是唯一必需的值，等差数列是产生 stop 的前一个值。例如，如果省略 start，所产生等差数列范围为 0 ～ stop-1。step 的预设是 1，所以预设等差数列是递增 1。如果将 step 设为 2，等差数列是递增 2。如果将 step 设为 -1，则产生递减的等差数列。

由 range( ) 产生的可迭代等差级数对象的数据类型是 range，可参考下列实例。

```
>>> x = range(3)
>>> type(x)
<class 'range'>
```

下列是打印 range( ) 对象内容。

```
>>> for x in range(3):
        print(x)


0
1
2
>>> for x in range(0,3):
        print(x)


0
1
2
```

上述执行循环迭代时，即使是执行 3 圈，但是系统不用一次预留 3 个整数空间存储循环计数指标，而是每次循环用 1 个整数空间存储循环计数指针，所以可以节省系统资源。下列是 range( ) 含 step 参数的应用，第 1 个是建立 1 ～ 10 的奇数序列，第 2 个是建立每次递减 2 的序列。

```
>>> for x in range(1,10,2):
        print(x)


1
3
5
7
9
>>> for x in range(3,-3,-2):
        print(x)


3
1
-1
```

7-2-1　只有一个参数的 range( ) 函数的应用

当 range(n) 函数搭配一个参数时：

```
range(n)        # 它将产生 0, 1, … , n-1 的可迭代对象内容
```

下列是测试 range( ) 方法。

程序实例 ch7_10.py：输入数字，本程序会将此数字当作打印星号的数量。

```
1  # ch7_10.py
2  n = int(input("请输入星号数量 : ")) # 定义星号的数量
3  for number in range(n):           # for循环
4      print("*",end="")             # 打印星号
```

执行结果

```
===================== RESTART: D:\Python\ch7\ch7_10.py =====================
请输入星号数量 : 3
***
>>>
===================== RESTART: D:\Python\ch7\ch7_10.py =====================
请输入星号数量 : 10
**********
```

7-2-2　扩充专题银行存款复利的轨迹

在 2-12 节有设计了银行复利的计算，当时由于 Python 所学语法有限所以无法看出每年本金和的变化，本节将以实例解说。

程序实例 ch7_11.py：参考 ch2_5.py 的利率与本金，以及年份，本程序会列出每年的本金和的轨迹。

```
1  # ch7_11.py
2  money = 50000
3  rate = 0.015
4  n = 5
5  for i in range(n):
6      money *= (1 + rate)
7      print("第 %d 年本金和 : %d" % ((i+1),int(money)))
```

执行结果

```
===================== RESTART: D:/Python/ch7/ch7_11.py =====================
第 1 年本金和 : 50749
第 2 年本金和 : 51511
第 3 年本金和 : 52283
第 4 年本金和 : 53068
第 5 年本金和 : 53864
```

7-2-3　有两个参数的 range( ) 函数

当 range( ) 函数搭配两个参数时，它的语法格式如下：

range(start, end) # **start 是起始值，end-1 是终止值**

上述语句可以产生 start 起始值到 end-1 终止值之间每次递增 1 的序列，start 或 end 可以是负整数，如果**终止值**小于**起始值**则是产生空序列或称空 range 对象，可参考下列程序实例。

```
>>> for x in range(10,2):
        print(x)


>>>
```

下列是使用负值当作起始值。

```
>>> for x in range(-1,2):
        print(x)


-1
0
1
```

程序实例 ch7_12.py：输入正整数值 n，这个程序会计算从 1 加到 n 的和。

```
1  # ch7_12.py
2  n = int(input("请输入n值 : "))
3  sum = 0
4  for num in range(1,n+1):
5      sum += num
6  print("总和 = ", sum)
```

执行结果

```
====================== RESTART: D:\Python\ch7\ch7_12.py ======================
请输入n值 : 10
总和 =  55
```

7-2-4　有 3 个参数的 range( ) 函数

当 range( ) 函数搭配 3 个参数时，它的语法格式如下：

range(start, end, step)　　　# start 是起始值，end 是终止值，step 是间隔值

然后会从起始值开始产生等差级数，每次间隔 step 时产生新数值元素，到 end-1 为止，下面是产生 2 ～ 10 的偶数。

```
>>> for x in range(2,11,2):
        print(x)


2
4
6
8
10
```

此外，step 值也可以是负值，此时起始值必须大于终止值。

```
>>> for x in range(10,0,-2):
        print(x)


10
8
6
4
2
```

7-2-5　活用 range( )

程序设计时也可以直接应用 range( )，让程序精简。

程序实例 ch7_13.py：输入一个正整数 n，这个程序会列出从 1 加到 n 的总和。

```
1  # ch7_13.py
2  n = int(input("请输入整数:"))
3  total = sum(range(n + 1))
4  print("从1到%d的总和是 = " % n, total)
```

执行结果

```
====================== RESTART: D:\Python\ch7\ch7_13.py ======================
请输入整数:10
从1到10的总和是 =  55
```

上述程序使用了可迭代对象的内建函数 sum 执行总和的计算，它的工作原理并不是一次预留存储 1, 2, …，10 的内存空间，然后执行运算，而是只有一个内存空间，每次将迭代的指标放在此空间，然后执行 sum( ) 运算，这样可以增加工作效率和节省系统内存空间。

程序实例 ch7_14.py：建立一个整数平方的列表，为了避免数值太大，若是输入大于 10，此大于 10 的数值将被设为 10。

```
1  # ch7_14.py
2  squares = []                      # 建立空列表
3  n = int(input("请输入整数:"))
4  if n > 10 : n = 10                # 最大值是10
5  for num in range(1, n+1):
6      value = num * num             # 元素平方
7      squares.append(value)         # 加入列表
8  print(squares)
```

执行结果

```
====================== RESTART: D:\Python\ch7\ch7_14.py ======================
请输入整数:12
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
>>>
====================== RESTART: D:\Python\ch7\ch7_14.py ======================
请输入整数:10
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
>>>
====================== RESTART: D:\Python\ch7\ch7_14.py ======================
请输入整数:5
[1, 4, 9, 16, 25]
```

对于上述程序而言，也可以使用 ** 代替**乘方**运算，同时第 6 和 7 行使用更精简的设计方式。

程序实例 ch7_15.py：用更精简方式设计 ch7_14.py。

```
1  # ch7_15.py
2  squares = []                      # 建立空列表
3  n = int(input("请输入整数:"))
4  if n > 10 : n = 10                # 最大值是10
5  for num in range(1, n+1):
6      squares.append(num ** 2)      # 加入列表
7  print(squares)
```

执行结果　与 ch7_14.py 相同。

7-2-6　删除列表内所有元素

程序实例 ch7_15_1.py：删除列表内所有元素。Python 没有提供删除整个列表元素的方法，不过可以使用 for 循环完成此工作。

```
1  # ch7_15_1.py
2  fruits = ['苹果', '香蕉', '西瓜', '水蜜桃', '百香果']
3  print("目前fruits列表 : ", fruits)
4
5  for fruit in fruits[:]:
6      fruits.remove(fruit)
7      print("删除 %s " % fruit)
8      print("目前fruits列表 : ", fruits)
```

执行结果

```
==================== RESTART: D:\Python\ch7\ch7_15_1.py ====================
目前fruits列表 :  ['苹果', '香蕉', '西瓜', '水蜜桃', '百香果']
删除 苹果
目前fruits列表 :  ['香蕉', '西瓜', '水蜜桃', '百香果']
删除 香蕉
目前fruits列表 :  ['西瓜', '水蜜桃', '百香果']
删除 西瓜
目前fruits列表 :  ['水蜜桃', '百香果']
删除 水蜜桃
目前fruits列表 :  ['百香果']
删除 百香果
目前fruits列表 :  []
```

7-2-7 列表生成的应用

生成式（generator）是一种使用迭代方式产生 Python 数据的方式，例如，可以产生列表、字典、集合等。这是结合**循环与条件表达式**的精简程序代码的方法，如果读者会用此概念设计程序，表示读者的 Python 功力已跳脱初学阶段，如果读者有其他程序语言经验，表示你已经逐渐跳脱其他程序语言的枷锁，蜕变成真正的 Python 程序设计师。

程序实例 ch7_15_2.py：建立 0 ～ 5 的列表，读者最初可能会用下列方法。

```
1  # ch7_15_2.py
2  xlst = []
3  xlst.append(0)
4  xlst.append(1)
5  xlst.append(2)
6  xlst.append(3)
7  xlst.append(4)
8  xlst.append(5)
9  print(xlst)
```

执行结果

```
==================== RESTART: D:/Python/ch7/ch7_15_2.py ====================
[0, 1, 2, 3, 4, 5]
```

如果要让程序设计更有效率，可以使用一个 for 循环和 range( ) 方法。

程序实例 ch7_15_3.py：使用一个 for 循环和 range( ) 方法重新设计上述程序。

```
1  # ch7_15_3.py
2  xlst = []
3  for n in range(6):
4      xlst.append(n)
5  print(xlst)
```

执行结果 与 ch7_15_2.py 相同。

或是直接使用 list( ) 将 range(n) 当作参数。

程序实例 ch7_15_4.py：直接使用 list( ) 将 range(n) 当作参数，重新设计上述程序。

```
1  # ch7_15_4.py
2  xlst = list(range(6))
3  print(xlst)
```

执行结果 与 ch7_15_3.py 相同。

上述方法均可以完成工作，但是如果要成为真正的 Python 工程师，建议使用列表生成式（list generator）。在说明实例前先看列表生成式的语法：

新列表 = [**表达式** for 项目 in 可迭代对象]

上述语法是，将每个可迭代对象套入**表达式**，每次产生一个列表元素。如果将列表生成式应用在上述实例中，整个内容如下：

xlst = [n for n in range(6)]

上述语句中第 1 个 n 是产生列表的值，也可以想成循环结果的值，第 2 个 n 是 for 循环的一部分，用于迭代 range(6) 的内容。

程序实例 ch7_15_5.py：用列表生成式产生列表。

```
1  # ch7_15_5.py
2  xlst = [ n for n in range(6)]
3  print(xlst)
```

执行结果 与 ch7_15_3.py 相同。

读者需记住，第 1 个 n 是产生列表的值，其实这部分也可以是一个表达式，

如果将上述语句应用于改良 ch7_15.py，可以将该程序第 5、6 行转成列表生成语法，此时内容可以修改如下：

square = [num ** 2 for num in range(1, n+1)]

此外，用这种方式设计时，可以省略第 2 行建立空列表。

程序实例 ch7_16.py：重新设计 ch7_15.py，进阶列表生成的应用。

```
1  # ch7_16.py
2  n = int(input("请输入整数:"))
3  if n > 10 : n = 10            # 最大值是10
4  squares = [num ** 2 for num in range(1, n+1)]
5  print(squares)
```

执行结果 与 ch7_15.py 相同。

程序实例 ch7_17.py：有一个摄氏温度列表 celsius，这个程序会利用此列表生成华氏温度列表 fahrenheit。

```
1  # ch7_17.py
2  celsius = [21, 25, 29]
3  fahrenheit = [(x * 9 / 5 + 32) for x in celsius]
4  print(fahrenheit)
```

执行结果

```
====================== RESTART: D:\Python\ch7\ch7_17.py ======================
[69.8, 77.0, 84.2]
```

程序实例 ch7_18.py：毕达哥拉斯直角三角形定义，即我们中学数学所学的勾股定理，基本概念是直角三角形两边长的平方和等于斜边的平方，如下。

$a^2 + b^2 = c^2$ # c 是斜边长

这个定理可以用（a, b, c）方式表达，最著名的实例是（3,4,5）。小括号是元组的表达方式，我们尚未介绍，所以本节使用 [a,b,c] 列表表示。这个程序会生成 0 ～ 19 中符合定义的 a、b、c 列表值。

```
1  # ch7_18.py
2  x = [[a, b, c] for a in range(1,20) for b in range(a,20) for c in range(b,20)
3        if a ** 2 + b ** 2 == c **2]
4  print(x)
```

执行结果

```
====================== RESTART: D:\Python\ch7\ch7_18.py ======================
[[3, 4, 5], [5, 12, 13], [6, 8, 10], [8, 15, 17], [9, 12, 15]]
```

程序实例 ch7_19.py：在数学的使用中可能会碰上下列数学定义。

A * B = {(a, b)}:a 属于 A 元素 ,b 属于 B 元素

可以用下列程序生成这类列表。

```
1  # ch7_19.py
2  colors = ["Red","Green","Blue"]
3  shapes = ["Circle","Square","Line"]
4  result = [[color,shape] for color in colors for shape in shapes]
5  print(result)
```

执行结果

```
====================== RESTART: D:\Python\ch7\ch7_19.py ======================
[['Red', 'Circle'], ['Red', 'Square'], ['Red', 'Line'], ['Green', 'Circle'], ['G
reen', 'Square'], ['Green', 'Line'], ['Blue', 'Circle'], ['Blue', 'Square'], ['B
lue', 'Line']]
```

7-2-8 打印含列表元素的列表

本节概念称为 list unpacking，这个程序会从每个列表中找出 color 和 shape 的列表元素值。

程序实例 ch7_20.py：简化上一个程序，然后列出列表内每个元素的列表值。

```
1  # ch7_20.py
2  colors = ["Red", "Green", "Blue"]
3  shapes = ["Circle", "Square"]
4  result = [[color, shape] for color in colors for shape in shapes]
5  for color, shape in result:
6      print(color, shape)
```

执行结果

```
======================= RESTART: D:/Python/ch7/ch7_20.py =======================
Red Circle
Red Square
Green Circle
Green Square
Blue Circle
Blue Square
```

7-2-9　含有条件式的列表生成

语法如下：

新列表 = [**表达式**　for　项目　in　可迭代对象　if　条件式]

下列是用传统方式建立 1, 3, … , 9 的列表。

```
>>> for num in range(1,10):
        if num % 2 == 1:
            oddlist.append(num)


>>> oddlist
[1, 3, 5, 7, 9]
```

下列是使用 Python 精神设计含有条件式的列表生成程序。

```
>>> oddlist = [num for num in range(1,10) if num % 2 == 1]
>>> oddlist
[1, 3, 5, 7, 9]
```

7-2-10　列出 ASCII 码值或 Unicode 码值的字符

学习程序语言重要是活用，在 3-5-1 节介绍了 ASCII 码，下面是列出码值为 32 ～ 127 的 ASCII 字符。

```
>>> for x in range(32,128):
        print(chr(x),end='')


 !"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmno
pqrstuvwxyz{|}~
```

在 3-5-2 节介绍了 Unicode，下面是产生 Unicode 字符 0x6d2a ～ 0x6e29。

```
>>> for x in range(0x6d2a, 0x6e2a):
        print(chr(x),end='')


洪洫洬洭洮洯洰洱洲洳洴洵洶洷洸洹洺活洼洽派洿浀流浂浃浄浅浆浇浈浉浊测浌浍济浏浐浑
浒浓浔浕浖浗浘浙浚浛浜浝浞浟浠浡浢浣浤浥浦浧浨浩浪浫浬浭浮浯浰浱浲浳浴浵浶海浸浹
浺浻浼浽浾浿涀涁涂涃涄涅涆涇消涉涊涋涌涍涎涏涐涑涒涓涔涕涖涗涘涙涚涛涜涝涞涟涠涡
涢涣涤涥润涧涨涩涪涫涬涭涮涯涰涱液涳涴涵涶涷涸涹涺涻涼涽涾涿淀淁淂淃淄淅淆淇淈淉
淊淋淌淍淎淏淐淑淒淓淔淕淖淗淘淙淚淛淜淝淞淟淠淡淢淣淤淥淦淧淨淩淪淫淬淭淮淯淰深
淲淳淴淵淶混淸淹淺添淼淽淾淿渀渁渂渃渄清渆渇済渉渊渋渌渍渎渏渐渑渒渓渔渕渖渗渘渙
渚減渜渝渞渟渠渡渢渣渤渥渦渧渨温
```

7-3 进阶的 for 循环应用

7-3-1 嵌套 for 循环

一个循环内有另一个循环，称为**嵌套循环**。如果外层循环要执行 n 次，内层循环要执行 m 次，则整个循环执行的次数是 n×m 次。设计这类循环时要特别注意下列事项。

（1）外层循环的索引值变量与内层循环的索引值变量不要相同，以免混淆。

（2）程序代码的内缩一定要小心。

下面是嵌套循环的基本语法。

```
for var1 in 可迭代对象 :                  # 外层 for 循环
…
    for var2 in 可迭代对象 :              # 内层 for 循环
         ….
```

程序实例 ch7_21.py：打印 9×9 的乘法表。

```
1  # ch7_21.py
2  for i in range(1, 10):
3      for j in range(1, 10):
4          result = i * j
5          print("%d*%d=%-3d" % (i, j, result), end=" ")
6      print()          # 换行输出
```

执行结果

```
====================== RESTART: D:/Python/ch7/ch7_21.py ======================
1*1=1   1*2=2   1*3=3   1*4=4   1*5=5   1*6=6   1*7=7   1*8=8   1*9=9
2*1=2   2*2=4   2*3=6   2*4=8   2*5=10  2*6=12  2*7=14  2*8=16  2*9=18
3*1=3   3*2=6   3*3=9   3*4=12  3*5=15  3*6=18  3*7=21  3*8=24  3*9=27
4*1=4   4*2=8   4*3=12  4*4=16  4*5=20  4*6=24  4*7=28  4*8=32  4*9=36
5*1=5   5*2=10  5*3=15  5*4=20  5*5=25  5*6=30  5*7=35  5*8=40  5*9=45
6*1=6   6*2=12  6*3=18  6*4=24  6*5=30  6*6=36  6*7=42  6*8=48  6*9=54
7*1=7   7*2=14  7*3=21  7*4=28  7*5=35  7*6=42  7*7=49  7*8=56  7*9=63
8*1=8   8*2=16  8*3=24  8*4=32  8*5=40  8*6=48  8*7=56  8*8=64  8*9=72
9*1=9   9*2=18  9*3=27  9*4=36  9*5=45  9*6=54  9*7=63  9*8=72  9*9=81
```

上述程序第 5 行中“%-3d”主要是供 result 使用，表示每一个输出预留 3 格，同时靠左输出。同一行中的 end=" " 则是设置输出完空一格，下次输出不换行输出。当内层循环执行完一次，则执行第 6 行，这是外层循环语句，主要是设置下次换行输出，相当于下次再执行内层循环时换行输出。

程序实例 ch7_22.py：绘制直角三角形。

```
1  # ch7_22.py
2  for i in range(1, 10):
3      for j in range(1, 10):
4          if j <= i:
5              print("aa", end="")
6      print()                  # 换行输出
```

执行结果

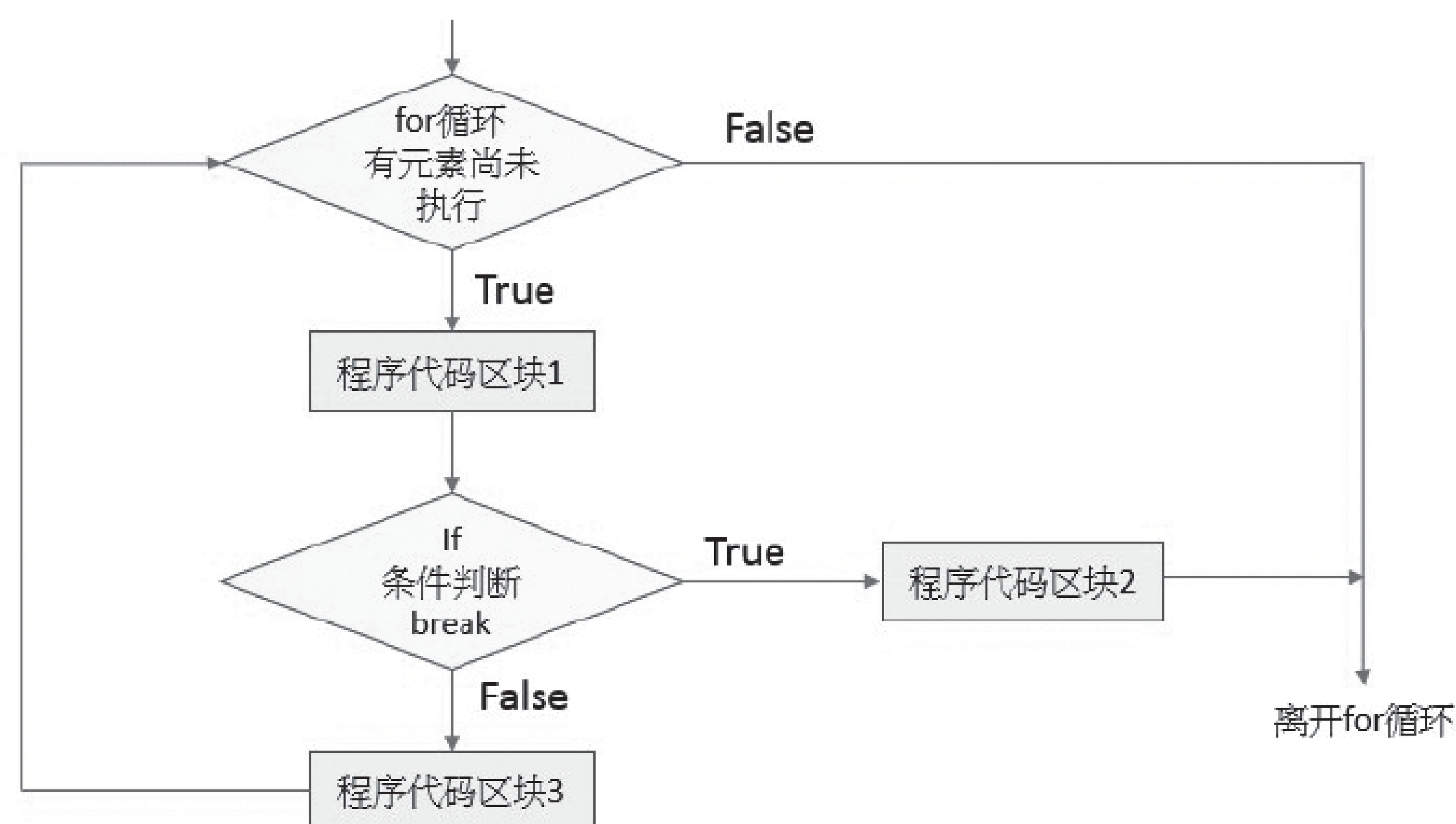
```
======================= RESTART: D:/Python/ch7/ch7_22.py =======================
aa
aaaa
aaaaaa
aaaaaaaa
aaaaaaaaaa
aaaaaaaaaaaa
aaaaaaaaaaaaaa
aaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaa
```

7-3-2　强制离开 for 循环——break 指令

在设计 for 循环时，如果期待某些条件发生时可以离开循环，可以在循环内执行 break 指令，即可立即离开循环，这个指令通常是和 if 语句配合使用。下面是常用的语法格式。

```
for var in 可迭代物件：
    程序代码区块 1
    if 条件表达式：                # 判断条件表达式
        程序代码区块 2
break                             # 如果条件表达式是 True 则离开 for 循环
    程序代码区块 3
```

下面是流程图，其中在 for 循环内的 if 条件判断，也许前方有程序代码区块 1、if 条件内有程序代码区块 2 或是后方有程序代码区块 3，只要 if 条件判断是 True，则执行 if 条件内的程序代码区块 2 后，可立即离开循环。



例如，你设计一个比赛，可以将参加比赛者的成绩列在列表内，如果想列出前 20 名参加决赛，可以设置 for 循环当选取 20 名后，即离开循环，此时就可以使用 break。

程序实例 ch7_23.py：输出一系列数字元素，当数字为 5 时，循环将终止执行。

```
1  # ch7_23.py
2  print("测试1")
3  for digit in range(1, 11):
4      if digit == 5:
5          break
6      print(digit, end=', ')
7  print()
8  print("测试2")
9  for digit in range(0, 11, 2):
10     if digit == 5:
11         break
12     print(digit, end=', ')
```

执行结果

```
====================== RESTART: D:\Python\ch7\ch7_23.py ======================
测试1
1, 2, 3, 4,
测试2
0, 2, 4, 6, 8, 10,
```

上述语句在第一个列表的测试中（第 3 ～ 6 行），当碰到列表元素是 5 时，循环将终止，所以只有列出“1, 2, 3, 4,”元素。在第二个列表的测试中（第 9 ～ 12 行），当碰到列表元素是 5 时，循环将终止，可是这个列表元素中没有 5，所以整个循环可以正常执行到结束。

程序实例 ch7_24.py：列出球员名称，列出多少个球员则是由屏幕输入，这个程序同时设置，如果屏幕输入的人数大于列表的球员数时，自动将所输入的人数降为列表的球员数。

```
1   # ch7_24.py
2   players = ['Curry', 'Jordan', 'James', 'Durant', 'Obama', 'Kevin', 'Lin']
3   n = int(input("请输入人数 = "))
4   if n > len(players) : n = len(players)  # 列出人数不大于列表元素数
5   index = 0                               # 索引
6   for player in players:
7       if index == n:
8           break
9       print(player, end=" ")
10      index += 1                          # 索引加1
```

执行结果

```
====================== RESTART: D:\Python\ch7\ch7_24.py ======================
请输入人数 = 5
Curry Jordan James Durant Obama
>>>
====================== RESTART: D:\Python\ch7\ch7_24.py ======================
请输入人数 = 9
Curry Jordan James Durant Obama Kevin Lin
```

程序实例 ch7_25.py：一个列表 scores 内含有 10 个分数元素，请列出最高分的前 5 个。

```
1   # ch7_25.py
2   scores = [94, 82, 60, 91, 88, 79, 61, 93, 99, 77]
3   scores.sort(reverse = True)         # 从大到小排列
4   count = 0
5   for sc in scores:
6       count += 1
7       print(sc, end=" ")
8       if count == 5:                  # 取前5名成绩
9           break                       # 离开for循环
```

执行结果

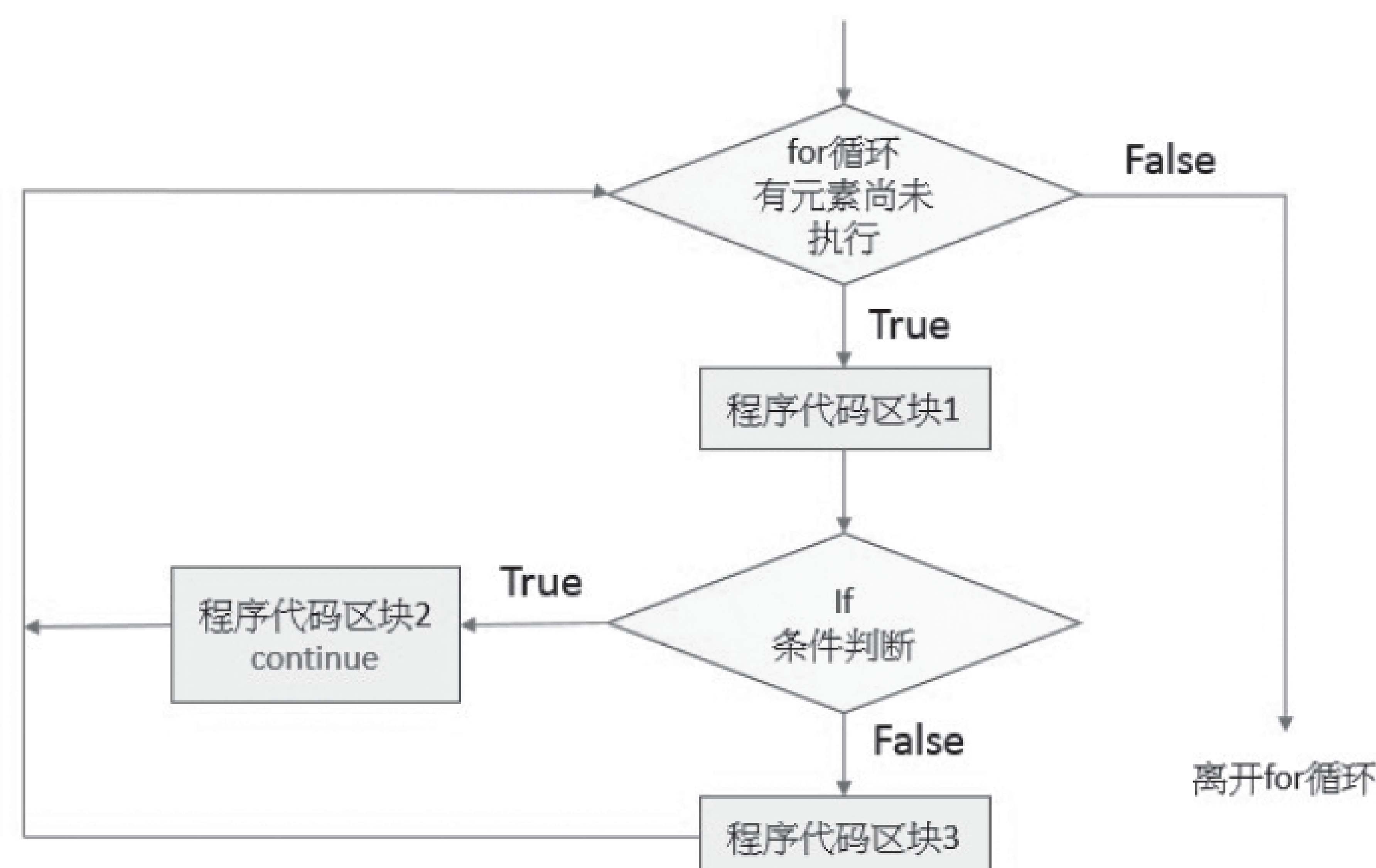
```
====================== RESTART: D:/Python/ch7/ch7_25.py ======================
99 94 93 91 88
```

7-3-3 for 循环暂时停止不往下执行——continue 指令

在设计 for 循环时，如果期待某些条件发生时可以不往下执行循环内容，此时可以用 continue 指令，这个指令通常和 if 语句配合使用。下面是常用的语法格式。

```
for var in 可迭代物件：
    程序代码区块 1
    if 条件表达式：          # 如果条件表达式是 True 则不执行程序代码区块 3
            程序代码区块 2
continue
    程序代码区块 3
```

下面是流程图，相当于如果发生 if 条件判断是 True 时，则不执行程序代码区块 3 内容。



程序实例 ch7_26.py：有一个列表 scores 记录 James 的比赛得分，设计一个程序可以列出 James 有多少场次得分大于或等于 30 分。

```
1  # ch7_26.py
2  scores = [33, 22, 41, 25, 39, 43, 27, 38, 40]
3  games = 0
4  for score in scores:
5      if score < 30:                  # 小于30则不往下执行
6          continue
7      games += 1                      # 场次加1
8  print("有%d场得分超过30分" % games)
```

执行结果

```
======================= RESTART: D:\Python\ch7\ch7_26.py =======================
有6场得分超过30分
```

程序实例 ch7_27.py：有一个列表 players，这个列表的元素也是列表，包含球员名字和身高数据，列出所有身高是 200（含）厘米以上的球员数据。

```
1   # ch7_27.py
2   players = [['James', 202],
3              ['Curry', 193],
4              ['Durant', 205],
5              ['Jordan', 199],
6              ['David', 211]]
7   for player in players:
8       if player[1] < 200:
9           continue
10      print(player)
```

执行结果

```
===================== RESTART: D:/Python/ch7/ch7_27.py =====================
['James', 202]
['Durant', 205]
['David', 211]
```

对于上述 for 循环而言，每次执行第 7 行时，player 的内容是 players 的一个元素，而这个元素是一个列表，例如，第一次执行时 player 内容如下：

```
['James', 202]
```

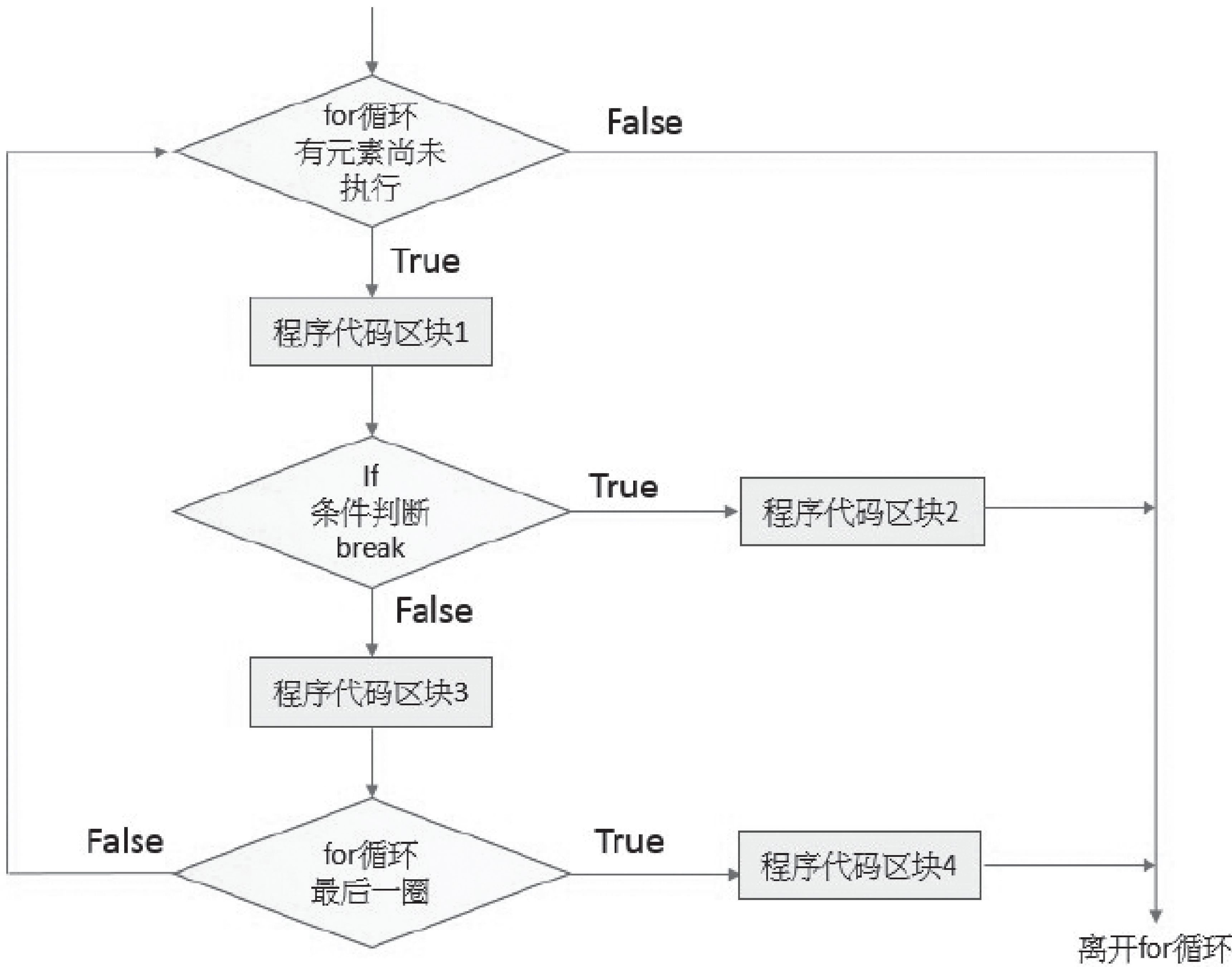
执行第 8 行时，player[1] 的值是 202。由于 if 判断的结果是 False，所以会执行第 10 行的 print(player) 指令，其他可以此类推。

7-3-4 for … else 循环

在设计 for 循环时，如果期待所有的 if 语句条件是 False 时，在最后一次循环后，可以执行特定程序区块指令，可使用这个语句，这个指令通常是和 if 和 break 语句配合使用的。下面是常用的语法格式。

```
for var in 可迭代对象：
    if 条件表达式：                # 如果条件表达式是 True 则离开 for 循环
            程序代码区块 1
break
    else:
    程序代码区块 2                 # 最后一次循环条件表达式是 False 则执行
```

下面是流程图，如果最后一次循环 if 条件表达式仍是 False 时，才会执行程序代码区块 2。



其实这个语法很适合传统数学中测试某一个数字 n 是否是**质数**（Prime Number），质数的条件是：

（1）2 是质数。

（2）n 不可被 2 ～ n-1 的数字整除。

程序实例 ch7_28.py：质数测试的程序，如果所输入的数字是质数则列出是质数，否则列出不是质数。

```
1  # ch7_28.py
2  num = int(input("请输入大于1的整数做质数测试 = "))
3  if num == 2:                                   # 2是质数所以直接输出
4      print("%d是质数" % num)
5  else:
6      for n in range(2, num):                    # 用2 .. num-1当除数测试
7          if num % n == 0:                       # 如果整除则不是质数
8              print("%d不是质数" % num)
9              break                              # 离开循环
10
11     else:                                      # 否则是质数
12         print("%d是质数" % num)
```

执行结果

```
====================== RESTART: D:\Python\ch7\ch7_28.py ======================
请输入大于1的整数做质数测试 = 2
2是质数
>>>
====================== RESTART: D:\Python\ch7\ch7_28.py ======================
请输入大于1的整数做质数测试 = 3
3是质数
>>>
====================== RESTART: D:\Python\ch7\ch7_28.py ======================
请输入大于1的整数做质数测试 = 12
12不是质数
```

7-4 while 循环

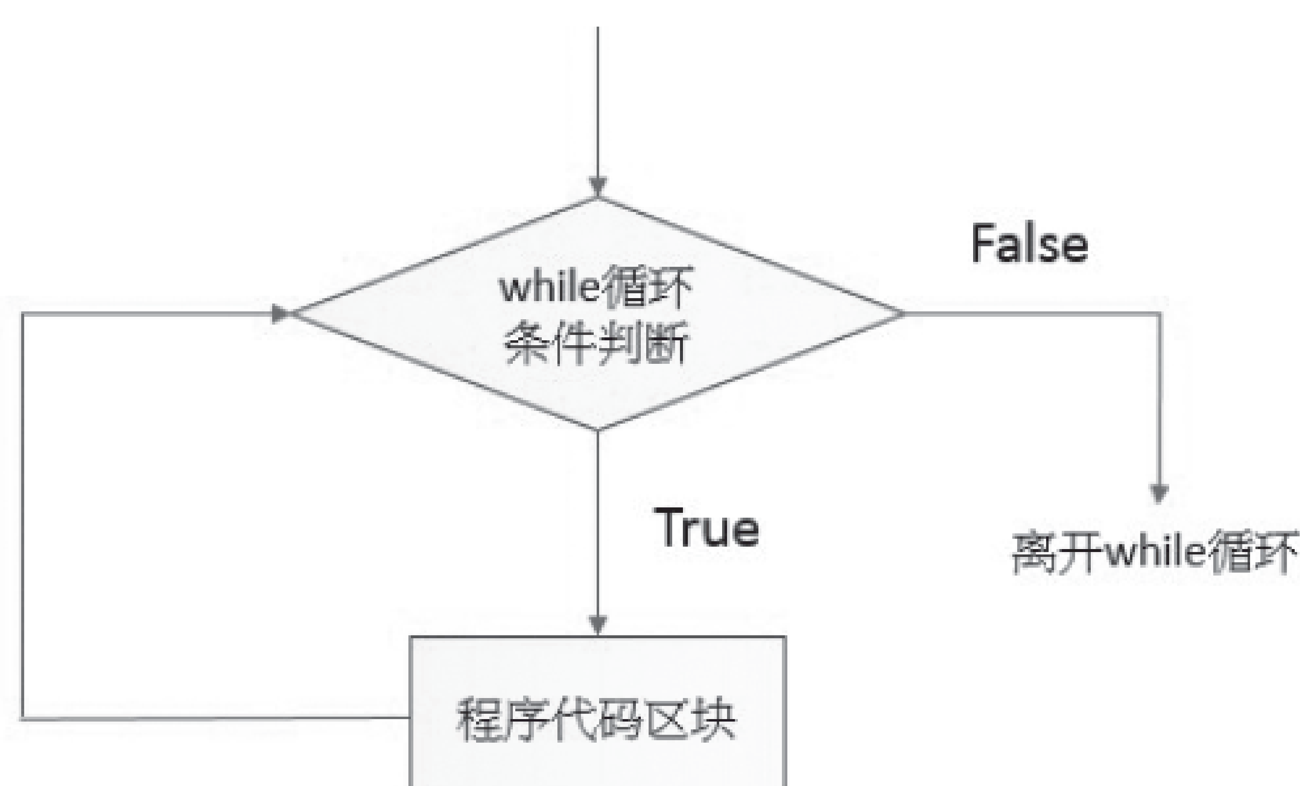
while 循环会一直执行直到条件运算为 False 时才会离开，所以设计 while 循环时一定要设计一个条件可以离开循环，相当于让循环结束。设计程序时，如果忘了设计条件可以离开循环，将造成无限循环状态，此时可以按 Ctrl+C 组合键，中断程序的执行离开无限循环的陷阱。

一般 while 循环使用的**语意是条件控制循环**，在符合特定条件下执行。for 循环则是一种**计数循环**，会重复执行特定次数。

while 条件运算 :

　　程序区块

下面是 while 循环语法流程图。



7-4-1　基本 while 循环

程序实例 ch7_29.py：这个程序会输出你所输入的内容，当输入 q 时，程序才会执行结束。

```
1  # ch7_29.py
2  msg1 = '人机对话专栏,告诉我心事吧,我会重复你告诉我的心事!'
3  msg2 = '输入 q 可以结束对话'
4  msg = msg1 + '\n' + msg2 + '\n' + '= '
5  input_msg = ''                  # 默认为空字符串
6  while input_msg != 'q':
7      input_msg = input(msg)
8      print(input_msg)
```

执行结果

```
====================== RESTART: D:\Python\ch7\ch7_29.py ======================
人机对话专栏,告诉我心事吧,我会重复你告诉我的心事!
输入 q 可以结束对话
= DeepMind
DeepMind
人机对话专栏,告诉我心事吧,我会重复你告诉我的心事!
输入 q 可以结束对话
= q
q
```

上述程序最大的缺点是，当输入 q 时，程序也将输出 q，然后才结束 while 循环，可以使用下列第 8 行增加 if 条件判断的方式改良。

程序实例 ch7_30.py：改良程序 ch7_29.py，当输入 q 时，不再输出 q。

```
1  # ch7_30.py
2  msg1 = '人机对话专栏,告诉我心事吧,我会重复你告诉我的心事!
3  msg2 = '输入 q 可以结束对话'
4  msg = msg1 + '\n' + msg2 + '\n' + '= '
5  input_msg = ''                    # 默认为空字符串
6  while input_msg != 'q':
7      input_msg = input(msg)
8      if input_msg != 'q':          # 如果输入不是q才输出信息
9          print(input_msg)
```

执行结果

```
====================== RESTART: D:\Python\ch7\ch7_30.py ======================
人机对话专栏,告诉我心事吧,我会重复你告诉我的心事!
输入 q 可以结束对话
= DeepMind
DeepMind
人机对话专栏,告诉我心事吧,我会重复你告诉我的心事!
输入 q 可以结束对话
= q
```

上述程序尽管可以完成工作，但是当我们在设计大型程序时，如果可以有更明确的标记，记录程序是否继续执行将更佳，下面将用一个布尔变量值 active 当作标记，如果是 True 则 while 循环继续，否则 while 循环结束。

程序实例 ch7_31.py：改良 ch7_30.py 程序的可读性，使用标记 active 记录是否循环继续。

```
1  # ch7_31.py
2  msg1 = '人机对话专栏,告诉我心事吧,我会重复你告诉我的心事!'
3  msg2 = '输入 q 可以结束对话'
4  msg = msg1 + '\n' + msg2 + '\n' + '= '
5  active = True
6  while active:                   # 循环进行直到active是False
7      input_msg = input(msg)
8      if input_msg != 'q':        # 如果输入不是q才输出信息
9          print(input_msg)
10     else:
11         active = False          # 输入是q所以将active设为False
```

执行结果　与 ch7_30.py 相同。

程序实例 ch7_32.py：猜数字游戏，程序第 2 行用变量 answer 存储要猜的数字，程序执行时用变量 guess 存储所猜的数字。

```
1  # ch7_32.py
2  answer = 30                    # 正确数字
3  guess = 0                      # 设置所猜数字的初始值
4  while guess != answer:
5      guess = int(input("请猜1-100间的数字 = "))
6      if guess > answer:
7          print("请猜小一点")
8      elif guess < answer:
9          print("请猜大一点")
10     else:
11         print("恭喜答对了")
```

执行结果

```
====================== RESTART: D:\Python\ch7\ch7_32.py ======================
请猜1-100间的数字 = 50
请猜小一点
请猜1-100间的数字 = 25
请猜大一点
请猜1-100间的数字 = 30
恭喜答对了
```

7-4-2　认识哨兵值

在程序设计时，可以在 while 循环中设置一个输入数值当作循环执行结束的值，这个值称为哨兵值（Sentinel Value）。

程序实例 ch7_33.py：计算输入值的总和，哨兵值是 0，如果输入 0 则程序结束。

```
1  # ch7_33.py
2  n = int(input("请输入一个值 : "))
3  sum = 0
4  while n != 0:
5      sum += n
6      n = int(input("请输入一个值 : "))
7  print("输入总和 = ", sum)
```

执行结果

```
====================== RESTART: D:\Python\ch7\ch7_33.py ======================
请输入一个值 : 5
请输入一个值 : 6
请输入一个值 : 7
请输入一个值 : 0
输入总和 =  18
```

7-4-3　预测学费

程序实例 ch7_34.py：假设今年大学学费是 5 万元，未来每年以 5% 速度向上涨价，计算多少年后学费会达到或超过 6 万元，学费不会少于 1 元，计算时忽略小数。

```
1  # ch7_34.py
2  tuition = 50000
3  year = 0
4  while tuition < 60000:
5      tuition = int(tuition * 1.05)
6      year += 1
7  print("经过 %d 年后学费会达到或超过60000元 " % year)
```

执行结果

```
====================== RESTART: D:\Python\ch7\ch7_34.py ======================
经过 4 年后学费会达到或超过60000元
```

7-4-4 嵌套 while 循环

while 循环也允许嵌套循环，此时的语法格式如下。

```
while 条件运算 :                    # 外层 while 循环
    …
    while 条件运算 :                # 内层 while 循环
        …
```

下面是我们已经知道的 while 循环会执行几次的应用。

程序实例 ch7_35.py：使用 while 循环重新设计 ch7_21.py，打印 9×9 乘法表。

```
1  # ch7_35.py
2  i = 1                       # 设置i初始值
3  while i <= 9:               # 当i大于9跳出外层循环
4      j = 1                   # 设置j初始值
5      while j <= 9:           # 当j大于9跳出内层循环
6          result = i * j
7          print("%d*%d=%-3d" % (i, j, result), end=" ")
8          j += 1              # 内层循环加1
9      print()                 # 换行输出
10     i += 1                  # 外层循环加1
```

执行结果 与 ch7_19.py 相同。

7-4-5 强制离开 while 循环——break 指令

7-3-2 节所介绍的 break 指令，也可以应用于 while 循环。在设计 while 循环时，如果期待某些条件发生时可以离开循环，可以在循环内执行 break 指令，即可立即离开循环，这个指令通常是和 if 语句配合使用。下面是常用的语法格式。

```
while 条件表达式 A:
    程序代码区块 1
    if 条件表达式 B:               # 判断条件表达式 A
        程序代码区块 2
break                              # 如果条件表达式 A 是 True 则离开 while 循环
    程序代码区块 3
```

程序实例 ch7_36.py：这个程序会先建立 while 无限循环，如果输入 q，则可跳出这个 while 无限循环。程序内容主要是要求输入水果名称，然后输出此水果。

```
1  # ch7_36.py
2  msg1 = '人机对话专栏,请告诉我妳喜欢吃的水果!'
3  msg2 = '输入 q 可以结束对话'
4  msg = msg1 + '\n' + msg2 + '\n' + '= '
5  while True:                          # 这是while无限循环
6      input_msg = input(msg)
7      if input_msg == 'q':             # 输入q可用break跳出循环
8          break
9      else:
10         print("我也喜欢吃 %s " % input_msg.title( ))
```

执行结果

```
===================== RESTART: D:\Python\ch7\ch7_36.py =====================
人机对话专栏,请告诉我妳喜欢吃的水果!
输入 q 可以结束对话
= apple
我也喜欢吃 Apple
人机对话专栏,请告诉我妳喜欢吃的水果!
输入 q 可以结束对话
= orange
我也喜欢吃 Orange
人机对话专栏,请告诉我妳喜欢吃的水果!
输入 q 可以结束对话
= q
```

程序实例 ch7_37.py：使用 while 循环重新设计 ch7_24.py。

```
1  # ch7_37.py
2  players = ['Curry', 'Jordan', 'James', 'Durant', 'Obama', 'Kevin', 'Lin']
3  n = int(input("请输入人数 = "))
4  if n > len(players) : n = len(players)   # 列出人数不大于列表元素数
5  index = 0                                  # 索引index
6  while index < len(players):                # index是否在列表长度范围
7      if index == n:                         # 是否达到想列出的人数
8          break
9      print(players[index], end=" ")
10     index += 1                             # 索引index加1
```

执行结果　与 ch7_24.py 相同。

上述程序第 6 行的“index < len(players)”相当于语法格式中的条件表达式 A，控制循环是否终止。程序第 7 行的 "index == n" 相当于语法格式中的条件表达式 B，可以控制是否中途离开 while 循环。

7-4-6　while 循环暂时停止——continue 指令

在设计 while 循环时，如果期待某些条件发生时可以不往下执行循环内容，此时可以用 continue 指令，这个指令通常是和 if 语句配合使用。下面是常用的语法格式。

```
while 条件运算 A:
    程序代码区块 1
    if 条件表达式 B:          # 如果条件表达式是 True 则不执行程序代码区块 3
        程序代码区块 2
```

    continue
        程序代码区块 3

程序实例 ch7_38.py：列出 1 ～ 10 的偶数。

```
1  # ch7_38.py
2  index = 0
3  while index <= 10:
4      index += 1
5      if ( index % 2 != 0 ):  # 测试是否奇数
6          continue            # 不往下执行
7      print(index)            # 输出偶数
```

执行结果

```
====================== RESTART: D:/Python/ch7/ch7_38.py ======================
2
4
6
8
10
```

7-4-7　while 循环条件表达式与可迭代对象

while 循环的条件表达式也可与可迭代对象配合使用，此时它的语法格式 1 如下。

while var in 可迭代对象：　　　# 如果 var in 可迭代对象是 True 则继续

程序区块

语法格式 2 如下。

while 可迭代：　　　　　　　　# 迭代对象是空的才结束

程序区块

程序实例 ch7_39.py：删除列表内的 apple 字符串元素，程序第 5 行表示只要在 fruits 列表内可以找到变量 fruit 内容是 apple，就会返回 True，循环将继续。

```
1  # ch7_39.py
2  fruits = ['apple', 'orange', 'apple', 'banana', 'apple']
3  fruit = 'apple'
4  print("删除前的fruits", fruits)
5  while fruit in fruits:      # 只要列表内有apple循环就继续
6      fruits.remove(fruit)
7  print("删除后的fruits", fruits)
```

执行结果

```
====================== RESTART: D:\Python\ch7\ch7_39.py ======================
删除前的fruits ['apple', 'orange', 'apple', 'banana', 'apple']
删除后的fruits ['orange', 'banana']
```

程序实例 ch7_40.py：有一个列表 buyers，此列表内含购买者和消费金额，如果购买金额超过或达到 1000 元，则归类为 VIP 买家 vipbuyers 列表，否则是 Gold 买家 goldbuyers 列表。

```
1  # ch7_40.py
2  buyers = [['James', 1030],                # 建立买家购买记录
3             ['Curry', 893],
4             ['Durant', 2050],
5             ['Jordan', 990],
6             ['David', 2110]]
7  goldbuyers = []                           # Gold买家列表
8  vipbuyers =[]                             # VIP买家列表
9  while buyers:                             # 执行买家分类循环分类完成循环才会结束
10     index_buyer = buyers.pop()
11     if index_buyer[1] >= 1000:            # 用1000元执行买家分类条件
12         vipbuyers.append(index_buyer)     # 加入VIP买家列表
13     else:
14         goldbuyers.append(index_buyer)    # 加入Gold买家列表
15 print("VIP 买家资料", vipbuyers)
16 print("Gold买家资料", goldbuyers)
```

执行结果

```
====================== RESTART: D:\Python\ch7\ch7_40.py ======================
VIP 买家资料 [['David', 2110], ['Durant', 2050], ['James', 1030]]
Gold买家资料 [['Jordan', 990], ['Curry', 893]]
```

上述程序第 9 行只要列表不是空列表，while 循环就会一直执行。

7-4-8　无限循环与 pass

pass 指令是什么事也不做，如果想要建立一个无限循环可以使用下列写法。

```
while  True:
    pass
```

也可以将 True 改为阿拉伯数字 1，如下所示。

```
while  1:
pass
```

不过不建议这么做，这会让程序进入无限循环。这个指令有时候会用于设计一个循环或函数（将在第 11-10 节解说）尚未完成时，先放 pass，未来再用完整程序代码取代。

程序实例 ch7_41.py：pass 应用于循环的实例，这个程序的循环尚未设计完成，所以先用 pass 处理。

```
1  # ch7_41.py
2  schools = ['明志科大', '台湾科大', '台北科大']
3  for school in schools:
4      pass
```

执行结果 没有任何数据输出。

7-5　enumerate 对象使用 for 循环解析

延续 6-12 节的 enumerate 对象可知，这个对象是由**索引值**与**元素值**配对出现。我们使用 for 循环迭代一般对象（例如列表）时，无法得知每个对象元素的索引，但是可以利用 enumerate( ) 方法建

立 enumerate 对象，建立原对象的索引信息。

然后可以使用 for 循环将每一个对象的**索引值与元素值**解析出来。

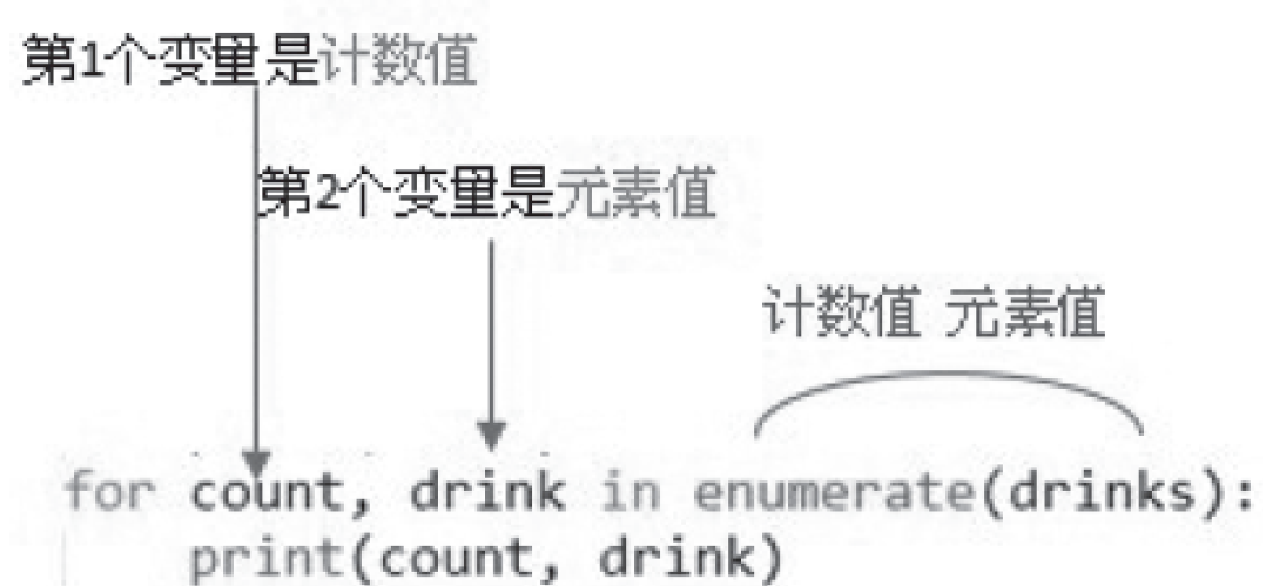
程序实例 ch7_42.py：继续设计 ch6_57.py，将 enumerate 对象的索引值与元素值解析出来。

```
1  # ch7_42.py
2  drinks = ["coffee", "tea", "wine"]
3  # 解析enumerate对象
4  for drink in enumerate(drinks):              # 数值初始是0
5      print(drink)
6  for count, drink in enumerate(drinks):
7      print(count, drink)
8  print("****************")
9  # 解析enumerate对象
10 for drink in enumerate(drinks, 10):          # 数值初始是10
11     print(drink)
12 for count, drink in enumerate(drinks, 10):
13     print(count, drink)
```

执行结果

```
===================== RESTART: D:/Python/ch7/ch7_42.py =====================
(0, 'coffee')
(1, 'tea')
(2, 'wine')
0 coffee
1 tea
2 wine
****************
(10, 'coffee')
(11, 'tea')
(12, 'wine')
10 coffee
11 tea
12 wine
```

上述程序第 6 行的思路如下。



由于 enumerate(drinks) 产生的 enumerate 对象是配对存在的，可以用两个变量遍历这个对象，只要仍有元素尚未被遍历循环就会继续。为了让读者了解 enumerate 对象的奥妙，笔者先用传统方式设计下列程序。

程序实例 ch7_43.py：以下是某位 NBA 球员的前 10 场的得分数据，可参考程序第 2 行，请用传统方式列出哪些场次得分超过 20 分（含）。注意：场次从第 1 场开始。

```
1  # ch7_43.py
2  scores = [21,29,18,33,12,17,26,28,15,19]
3  # 不使用enumerate对象
4  index = 1
5  for score in scores:
6      if score >= 20:
7          print("场次 %d : 得分 %d " % (index, score))
8      index += 1
```

执行结果

```
====================== RESTART: D:\Python\ch7\ch7_43.py ======================
场次 1 : 得分 21
场次 2 : 得分 29
场次 4 : 得分 33
场次 7 : 得分 26
场次 8 : 得分 28
```

请留意上述程序，我们必须建立索引变量与设置此索引的初值，可参考第 4 行，然后每次迭代时必须在第 8 行为索引增加 1。如果读者懂得 enumerate( ) 的意义，可以用下列程序轻松有效率地处理上述问题。

程序实例 ch7_44.py：使用 enumerate( ) 重新设计 ch7_43.py。

```
1  # ch7_44.py
2  scores = [21,29,18,33,12,17,26,28,15,19]
3  # 解析enumerate物件
4  for count, score in enumerate(scores, 1):    # 数值初始是 1
5      if score >= 20:
6          print("场次 %d : 得分 %d " % (count, score))
```

执行结果 与 ch7_43.py 相同。

其实一个人是不是 Python 高手，可以用上述问题测试，使用 ch7_44.py 方式设计才算是真正懂 Python 的高手。

7-6 专题——购物车设计 / 成绩系统 / 圆周率

7-6-1 设计购物车系统

程序实例 ch7_45.py：简单购物车的设计，这个程序执行时会列出所有商品，读者可以选择商品，如果所输入商品在商品列表则加入购物车，如果输入 Q 或 q 则购物结束，输出所购买商品。

```
1  # ch7_45.py
2  store = 'DeepStone购物中心'
3  products = ['电视','冰箱','洗衣机','电扇','冷气机']
4  cart = []                       # 购物车
5  print(store)
6  print(products,"\n")
7  while True:                     # 这是while无限循环
8      msg = input("请输入购买商品(q=quit) : ")
9      if msg == 'q' or msg=='Q':
10         break
11     else:
12         if msg in products:
13             cart.append(msg)
14
15 print("今天购买商品", cart)
```

执行结果

```
===================== RESTART: D:\Python\ch7\ch7_45.py =====================
DeepStone购物中心
['电视', '冰箱', '洗衣机', '电扇', '冷气机']

请输入购买商品(q=quit) : 冰箱
请输入购买商品(q=quit) : q
今天购买商品 ['冰箱']
```

7-6-2　建立真实的成绩系统

在 6-7-3 节介绍了成绩系统的计算，如下所示。

姓名	语文	英文	数学	总分
洪锦魁	80	95	88	0
洪冰儒	98	97	96	0
洪雨星	91	93	95	0
洪冰雨	92	94	90	0
洪星宇	92	97	80	0

其实更真实的成绩系统应该如下所示。

座号	姓名	语文	英文	数学	总分	平均	名次
1	洪锦魁	80	95	88	0	0	0
2	洪冰儒	98	97	96	0	0	0
3	洪雨星	91	93	95	0	0	0
4	洪冰雨	92	94	90	0	0	0
5	洪星宇	92	97	80	0	0	0

在上述成绩系统表格中，使用各科考试成绩然后必须填入每个人的总分、平均、名次。要处理上述成绩系统，关键是学会二维列表的排序，如果想针对列表内第 n 个元素值排序，可使用如下方法。

二维列表 .sort(key=lambda x:x[n])

上述函数方法参数有 lambda 关键词，读者可以不理会直接参考输入，即可获得排序结果，未来介绍函数时，在 11-9 节会介绍此关键词。

程序实例 ch7_46.py：设计真实的成绩系统排序。

```
1  # ch7_46.py
2  sc = [[1, '洪锦魁', 80, 95, 88, 0, 0, 0],
3        [2, '洪冰儒', 98, 97, 96, 0, 0, 0],
4        [3, '洪雨星', 91, 93, 95, 0, 0, 0],
5        [4, '洪冰雨', 92, 94, 90, 0, 0, 0],
6        [5, '洪星宇', 92, 97, 80, 0, 0, 0],
7       ]
8  # 计算总分与平均
9  print("填入总分与平均")
10 for i in range(len(sc)):
11     sc[i][5] = sum(sc[i][2:5])              # 填入总分
12     sc[i][6] = round((sc[i][5] / 3), 1)     # 填入平均
13     print(sc[i])
14 sc.sort(key=lambda x:x[5],reverse=True)     # 依据总分高往低排序
15 # 以下填入名次
16 print("填入名次")
17 for i in range(len(sc)):                    # 填入名次
18     sc[i][7] = i + 1
19     print(sc[i])
20 # 以下依座号排序
21 sc.sort(key=lambda x:x[0])                  # 依据座号排序
22 print("最后成绩单")
23 for i in range(len(sc)):
24     print(sc[i])
```

执行结果

```
===================== RESTART: D:\Python\ch7\ch7_46.py =====================
填入总分与平均
[1, '洪锦魁', 80, 95, 88, 263, 87.7, 0]
[2, '洪冰儒', 98, 97, 96, 291, 97.0, 0]
[3, '洪雨星', 91, 93, 95, 279, 93.0, 0]
[4, '洪冰雨', 92, 94, 90, 276, 92.0, 0]
[5, '洪星宇', 92, 97, 80, 269, 89.7, 0]
填入名次
[2, '洪冰儒', 98, 97, 96, 291, 97.0, 1]
[3, '洪雨星', 91, 93, 95, 279, 93.0, 2]
[4, '洪冰雨', 92, 94, 90, 276, 92.0, 3]
[5, '洪星宇', 92, 97, 80, 269, 89.7, 4]
[1, '洪锦魁', 80, 95, 88, 263, 87.7, 5]
最后成绩单
[1, '洪锦魁', 80, 95, 88, 263, 87.7, 5]
[2, '洪冰儒', 98, 97, 96, 291, 97.0, 1]
[3, '洪雨星', 91, 93, 95, 279, 93.0, 2]
[4, '洪冰雨', 92, 94, 90, 276, 92.0, 3]
[5, '洪星宇', 92, 97, 80, 269, 89.7, 4]
```

我们成功地建立了成绩系统，但是上述成绩系统还不是完美，如果两个人的成绩相同，座号属于后面的人名次将往下掉一名。

程序实例 ch7_47.py：修改成绩报告，如下所示。

座号	姓名	语文	英文	数学	总分	平均	名次
1	洪锦魁	80	95	88	0	0	0
2	洪冰儒	98	97	96	0	0	0
3	洪雨星	91	93	95	0	0	0
4	洪冰雨	92	94	90	0	0	0
5	洪星宇	92	97	90	0	0	0

请注意洪星宇的数学成绩是 90 分，下列是程序实例 ch7_47.py 的执行结果。

```
===================== RESTART: D:\Python\ch7\ch7_47.py =====================
填入总分与平均
[1, '洪锦魁', 80, 95, 88, 263, 87.7, 0]
[2, '洪冰儒', 98, 97, 96, 291, 97.0, 0]
[3, '洪雨星', 91, 93, 95, 279, 93.0, 0]
[4, '洪冰雨', 92, 94, 90, 276, 92.0, 0]
[5, '洪星宇', 92, 97, 90, 279, 93.0, 0]
填入名次
[2, '洪冰儒', 98, 97, 96, 291, 97.0, 1]
[3, '洪雨星', 91, 93, 95, 279, 93.0, 2]
[5, '洪星宇', 92, 97, 90, 279, 93.0, 3]
[4, '洪冰雨', 92, 94, 90, 276, 92.0, 4]
[1, '洪锦魁', 80, 95, 88, 263, 87.7, 5]
最后成绩单
[1, '洪锦魁', 80, 95, 88, 263, 87.7, 5]
[2, '洪冰儒', 98, 97, 96, 291, 97.0, 1]
[3, '洪雨星', 91, 93, 95, 279, 93.0, 2]
[4, '洪冰雨', 92, 94, 90, 276, 92.0, 4]
[5, '洪星宇', 92, 97, 90, 279, 93.0, 3]
```

很明显洪星宇与洪雨星总分相同，但是洪星宇的座号比较靠后造成名次是第 3 名，相同成绩的洪雨星是第 2 名。要解决这类问题有两个方法，一是在填入名次时检查分数是否和前一个分数相同，如果相同则采用前一个序列的名次；另一个方法是在填入名次后增加一个循环，检查是否有成绩总分相同，相当于每个总分与前一个总分做比较，如果与前一个总分相同，必须将名次调整为与前一个元素名次相同，这将是读者的习题。

7-6-3　计算圆周率

在第 2 章的习题 7 中介绍了计算圆周率的知识，笔者使用了**莱布尼茨**公式，当时也说明了此级

数收敛速度很慢，本节将用循环处理这类问题。可以用下列公式说明莱布尼茨公式。

$$pi = 4\left(1 - \frac{1}{3} + \frac{1}{5} + \frac{1}{7} + \cdots + \frac{(-1)^{i+1}}{2i-1}\right)$$

程序实例 ch7_48.py：使用莱布尼茨公式计算圆周率，这个程序会计算到 100 万次，同时每 10 万次列出一次圆周率的计算结果。

```
1  # ch7_48.py
2  x = 1000001
3  pi = 0
4  for i in range(1,x+1):
5      pi += 4*((-1)**(i+1) / (2*i-1))
6      if i != 1 and i % 100000 == 0:      # 隔100000执行一次
7          print("当 i = %7d 时 PI = %20.19f" % (i, pi))
```

执行结果

```
====================== RESTART: D:\Python\ch7\ch7_48.py ======================
当 i =  100000 时 PI = 3.1415826535897197758
当 i =  200000 时 PI = 3.1415876535897617750
当 i =  300000 时 PI = 3.1415893202564642017
当 i =  400000 时 PI = 3.1415901535897439167
当 i =  500000 时 PI = 3.1415906535896920282
当 i =  600000 时 PI = 3.1415909869230147500
当 i =  700000 时 PI = 3.1415912250182609355
当 i =  800000 时 PI = 3.1415914035897172241
当 i =  900000 时 PI = 3.1415915424786509114
当 i = 1000000 时 PI = 3.1415916535897743245
```

从上述结果可以得到当循环到 40 万次后，此圆周率才进入我们熟知的 3.14159xx。

习题

1. 有一列表内部的元素是一系列图文件，如下所示。(7-1 节)

da1.jpg、da2.png、da3.gif、da4.gif、da5.jpg、da6.jpg、da7.gif

请将 ".jpg"".png"".gif" 分别放置在 jpg、png、gif 列表，然后打印这些列表。

```
======================= RESTART: D:/Python/ex/ex7_1.py =======================
jpg文件列表 ['da1.jpg', 'da5.jpg', 'da6.jpg']
png文件列表 ['da2.png']
gif文件列表 ['da3.gif', 'da4.gif', 'da7.gif']
```

2. 有一个列表 players，这个列表的元素也是列表，包含球员名字和身高数据，['James', 202]、['Curry', 193]、['Durant', 205]、['Joradn', 199]、['David', 211]，列出所有身高是 200（含）厘米以上的球员数据。(7-1 节)

```
======================= RESTART: D:/Python/ex/ex7_2.py =======================
['James', 202]
['Durant', 205]
['David', 211]
```

3. 扩充程序 ch7_11.py，请将本金、年利率与存款年数从屏幕输入。(7-2 节)

```
======================= RESTART: D:/Python/ex/ex7_3.py =======================
请输入存款本金 : 50000
请输入年利率    : 0.015
请输入多少年    : 5
第 1 年本金和 : 50749
第 2 年本金和 : 51511
第 3 年本金和 : 52283
第 4 年本金和 : 53068
第 5 年本金和 : 53864
```

4. 假设你今年体重是 50 千克，每年可以增加 1.2 千克，请列出未来 5 年的体重变化。(7-2 节)

```
======================= RESTART: D:/Python/ex/ex7_4.py =======================
第 1 年体重 : 51.2
第 2 年体重 : 52.4
第 3 年体重 : 53.6
第 4 年体重 : 54.8
第 5 年体重 : 56.0
```

5. 请使用 for 循环执行下列工作，输入 n 和 m 整数值，m 值一定大于 n 值，请列出 n 加到 m 的结果。例如，假设输入 n 值是 1，m 值是 100，则程序必须列出 1 加到 100 的结果是 5050。(7-2 节)

```
======================= RESTART: D:/Python/ex/ex7_5.py =======================
请输入n值 : 1
请输入m值 : 10
结果 =  55
>>>
======================= RESTART: D:/Python/ex/ex7_5.py =======================
请输入n值 : 10
请输入m值 : 15
结果 =  75
```

6. 有一个华氏温度列表 fahrenheit 内容是 [32, 77, 104]，这个程序会利用此列表产生摄氏温度列表 celsius。(7-2 节)

```
======================= RESTART: D:/Python/ex/ex7_6.py =======================
[0.0, 25.0, 40.0]
```

7. 参考 7-2-7 节产生 2,4,6, …，20 的列表。(7-2 节)

```
======================= RESTART: D:/Python/ex/ex7_7.py =======================
[2, 4, 6, 8, 10, 12, 14, 16, 18, 20]
```

8. 编写数字 1 ～ 5 中两个数字的各种组合。(7-2 节)

```
======================= RESTART: D:/Python/ex/ex7_8.py =======================
[[1, 1], [1, 2], [1, 3], [1, 4], [1, 5], [2, 1], [2, 2], [2, 3], [2, 4], [2, 5],
 [3, 1], [3, 2], [3, 3], [3, 4], [3, 5], [4, 1], [4, 2], [4, 3], [4, 4], [4, 5],
 [5, 1], [5, 2], [5, 3], [5, 4], [5, 5]]
```

9. 计算数学常数 e 值，它的全名是 Euler's number，又称欧拉数，主要是为了纪念瑞士数学家欧拉，这是一个无限不循环小数，可以使用下列级数计算 e 值。

$$e = 1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \cdots + \frac{1}{i!}$$

这个程序会计算到 i=100，同时每隔 10，列出一次计算结果。(7-2 节)

```
======================= RESTART: D:\Python\ex\ex7_9.py =======================
当i是  10 时 e = 2.7182818011463845131459038384491577744448
当i是  20 时 e = 2.7182818284590455348848081484902650117787
当i是  30 时 e = 2.7182818284590455348848081484902650117787
当i是  40 时 e = 2.7182818284590455348848081484902650117787
当i是  50 时 e = 2.7182818284590455348848081484902650117787
当i是  60 时 e = 2.7182818284590455348848081484902650117787
当i是  70 时 e = 2.7182818284590455348848081484902650117787
当i是  80 时 e = 2.7182818284590455348848081484902650117787
当i是  90 时 e = 2.7182818284590455348848081484902650117787
当i是 100 时 e = 2.7182818284590455348848081484902650117787
```

10. 请重新设计 ch7_22.py，输出更改为“1,2,…，9”，但是要得到下列结果。(7-2 节)

```
===================== RESTART: D:/Python/ex/ex7_10.py =====================
123456789
12345678
1234567
123456
12345
1234
123
12
1
```

11. 请重新设计 ch7_22.py，输出更改为“1,2,…，9”，但是要得到下列结果。(7-2 节)

```
===================== RESTART: D:/Python/ex/ex7_11.py =====================
        1
       21
      321
     4321
    54321
   654321
  7654321
 87654321
987654321
```

12. 列出 9×9 乘法表，其中标题输出需使用 center( ) 方法。(7-3 节)

```
===================== RESTART: D:/Python/ex/ex7_12.py =====================
      9 * 9 Multiplication Table
   1   2   3   4   5   6   7   8   9
=======================================
1 |  1   2   3   4   5   6   7   8   9
2 |  2   4   6   8  10  12  14  16  18
3 |  3   6   9  12  15  18  21  24  27
4 |  4   8  12  16  20  24  28  32  36
5 |  5  10  15  20  25  30  35  40  45
6 |  6  12  18  24  30  36  42  48  54
7 |  7  14  21  28  35  42  49  56  63
8 |  8  16  24  32  40  48  56  64  72
9 |  9  18  27  36  45  54  63  72  81
```

13. 计算前 20 个质数，然后放在列表中同时打印此列表。(7-4 节)

```
===================== RESTART: D:/Python/ex/ex7_13.py =====================
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71]
```

14. 扩充 ch7_32.py，增加列出所猜次数。(7-4 节)

```
===================== RESTART: D:\Python\ex\ex7_14.py =====================
请猜1-100间的数字 = 50
请猜小一点
请猜1-100间的数字 = 20
请猜大一点
请猜1-100间的数字 = 30
恭喜答对了
共猜 3 次
```

15. 扩充设计 ch7_40.py，有一个列表 buyers，此列表内含购买者和消费金额，若是购买金额达到 10000 元或以上，归类为 infinitebuyers 列表；如果购买金额超过或达到 1000 元，则归类为 VIP 买家 vipbuyers 列表；否则是 Gold 买家 goldbuyers 列表。此程序的原始列表数据如下。(7-4 节)

```
buyers = [['James', 1030],
['Curry', 893],
['Durant', 2050],
['Jordan', 990],
['David', 2110],
          ['Kevin', 15000],
```

```
        ['Mary', 10050],
        ['Tom', 8800],
    ]
```

```
===================== RESTART: D:\Python\ex\ex7_15.py =====================
Infinite 买家资料 [['Mary', 10050], ['Kevin', 15000]]
VIP 买家资料 [['Tom', 8800], ['David', 2110], ['Durant', 2050], ['James', 1030]]
Gold买家资料 [['Jordan', 990], ['Curry', 893]]
```

16. 请输入两个数，这个程序会求这两个数值的**最大公约数**（Greatest Common Divisor，GCD）。所谓的公约数是指可以被两个数字整除的数字，最大公约数是指可以被两个数字整除的最大值。例如，16 和 40 的公约数有 1、2、4、8，其中 8 就是最大公约数。（7-4 节）

```
===================== RESTART: D:\Python\ex\ex7_16.py =====================
请输入数值 1 : 16
请输入数值 2 : 40
16 和 40 的最大公约数是 : 8
>>>
===================== RESTART: D:\Python\ex\ex7_16.py =====================
请输入数值 1 : 99
请输入数值 2 : 33
99 和 33 的最大公约数是 : 33
```

17. 有一个水果列表如下：（7-5 节）

```
    fruits = ['李子', '香蕉', '苹果', '西瓜', '桃子']
```

请用含编号方式列出这些水果。

```
===================== RESTART: D:\Python\ex\ex7_17.py =====================
1 : 李子
2 : 香蕉
3 : 苹果
4 : 西瓜
5 : 桃子
```

18. 请修正 7-6 节的成绩系统，当总分相同时名次应该相同，这个作业需列出原始成绩单与最后成绩单。（7-6 节）

```
===================== RESTART: D:\Python\ex\ex7_18.py =====================
原始成绩单
[1, '洪锦魁', 80, 95, 88, 0, 0, 0]
[2, '洪冰儒', 98, 97, 96, 0, 0, 0]
[3, '洪雨星', 91, 93, 95, 0, 0, 0]
[4, '洪冰雨', 92, 94, 90, 0, 0, 0]
[5, '洪星宇', 92, 97, 90, 0, 0, 0]
最后成绩单
[1, '洪锦魁', 80, 95, 88, 263, 87.7, 5]
[2, '洪冰儒', 98, 97, 96, 291, 97.0, 1]
[3, '洪雨星', 91, 93, 95, 279, 93.0, 2]
[4, '洪冰雨', 92, 94, 90, 276, 92.0, 4]
[5, '洪星宇', 92, 97, 90, 279, 93.0, 2]
```

08

第 8 章

元组

本章摘要

在大型的商业或游戏网站设计中，列表（list）是非常重要的数据类型，因为记录各种等级客户、游戏角色等，都需要使用列表，**列表数据可以随时变动更新**。Python 提供另一种数据类型，称**元组**（tuple），这种数据类型结构与列表完全相同，与列表最大的差异是，它的**元素值与元素个数不可改动**，有时又可称为**不可改变的列表**，这也是本章的主题。

8-1 元组的定义

列表在定义时是将元素放在中括号内，元组的定义则是将元素放在**小括号**“( )”内，下列是元组的语法格式。

```
name_tuple = ( 元素 1, 元素 2,…  ,  元素 n, )      # name_tuple 是假设的元组名称
```

元组中的每一个数据称为**元素**，元素可以是**整数**、**字符串**或**列表**等，这些元素放在小括号 ( ) 内，彼此用逗号“,”隔开，最右边的元素 n 后的“,”可有可无。如果要打印元组内容，可以使用 print( ) 函数，将**元组名称**当作**变量名称**即可。

如果元组内的元素只有一个，在定义时需在元素右边加上逗号（“,”）。

```
name_tuple = ( 元素 1, )                    # 只有一个元素的元组
```

程序实例 ch8_1.py：定义与打印元组，最后使用 type( ) 列出元组数据类型。

```
1  # ch8_1.py
2  numbers1 = (1, 2, 3, 4, 5)       # 定义元组元素是整数
3  fruits = ('apple', 'orange')     # 定义元组元素是字符串
4  mixed = ('James', 50)            # 定义元组元素是不同类型数据
5  val_tuple = (10,)                # 只有一个元素的元组
6  print(numbers1)
7  print(fruits)
8  print(mixed)
9  print(val_tuple)
10 # 列出元组数据类型
11 print("元组mixed数据类型是: ",type(mixed))
```

执行结果

```
====================== RESTART: D:\Python\ch8\ch8_1.py ======================
(1, 2, 3, 4, 5)
('apple', 'orange')
('James', 50)
(10,)
元组mixed数据类型是:  <class 'tuple'>
```

另外一个简便建立有多个元素的元组的方法是用等号，右边有一系列元素，元素彼此用逗号隔开。

实例：简便建立元组的方法。

```
>>> x = 5, 6
>>> type(x)
<class 'tuple'>
>>> x
(5, 6)
```

8-2 读取元组元素

定义元组时是使用小括号“( )”，如果想要读取元组内容和列表可以用**中括号**“[ ]”。在 Python 中元组元素是从索引值 0 开始的，所以如果是元组的第一个元素，索引值是 0，第二个元素索引值是 1，其他以此类推，如下所示。

```
name_tuple[i]                    # 读取索引 i 的元组元素
```

程序实例 ch8_2.py：读取元组元素，一次指定多个变量值。

```
1  # ch8_2.py
2  numbers1 = (1, 2, 3, 4, 5)      # 定义元组元素是整数
3  fruits = ('apple', 'orange')    # 定义元组元素是字符串
4  val_tuple = (10,)               # 只有一个元素的元组
5  print(numbers1[0])              # 以中括号索引值读取元素内容
6  print(numbers1[4])
7  print(fruits[0],fruits[1])
8  print(val_tuple[0])
9  x, y = ('apple', 'orange')      # 有趣的应用也可以用x,y=fruits
10 print(x,y)
```

执行结果

```
===================== RESTART: D:\Python\ch8\ch8_2.py =====================
1
5
apple orange
10
apple orange
```

8-3 遍历所有元组元素

在 Python 中可以使用 for 循环遍历所有元组元素，用法与列表相同。

程序实例 ch8_3.py：假设元组是由字符串和数值组成，这个程序会列出元组的所有元素内容。

```
1  # ch8_3.py
2  keys = ('magic', 'xaab', 9099)      # 定义元组元素是字符串与数字
3  for key in keys:
4      print(key)
```

执行结果

```
===================== RESTART: D:\Python\ch8\ch8_3.py =====================
magic
xaab
9099
```

8-4 修改元组内容产生错误的实例

本章前言已经说明元组元素内容是不可更改的，下列是尝试更改元组元素内容的错误实例。

程序实例 ch8_4.py：修改元组内容产生错误的实例。

```
1  # ch8_4.py
2  fruits = ('apple', 'orange')          # 定义元组元素是字符串
3  print(fruits[0])                      # 打印元组fruits[0]
4  fruits[0] = 'watermelon'              # 将元素内容改为watermelon
5  print(fruits[0])                      # 打印元组fruits[0]
```

执行结果 下列是列出错误的画面。

```
====================== RESTART: D:\Python\ch8\ch8_4.py ======================
apple
Traceback (most recent call last):
  File "D:\Python\ch8\ch8_4.py", line 4, in <module>
    fruits[0] = 'watermelon'              # 将元素内容改为watermelon
TypeError: 'tuple' object does not support item assignment
```

上述出现错误消息，指出第 4 行错误，TypeError 指出 tuple 对象不支持赋值，相当于不可更改它的元素值。

8-5 使用全新定义方式修改元组元素

如果想修改元组元素，可以使用重新定义元组的方式处理。

程序实例 ch8_5.py：用重新定义方式修改元组元素内容。

```
1  # ch8_5.py
2  fruits = ('apple', 'orange')          # 定义元组元素是水果
3  print("原始fruits元组元素")
4  for fruit in fruits:
5      print(fruit)
6
7  fruits = ('watermelon', 'grape')      # 定义新的元组元素
8  print("\n新的fruits元组元素")
9  for fruit in fruits:
10     print(fruit)
```

执行结果

```
====================== RESTART: D:\Python\ch8\ch8_5.py ======================
原始fruits元组元素
apple
orange

新的fruits元组元素
watermelon
grape
```

8-6 元组切片

元组切片的概念与 6-1-3 节列表切片概念相同，下面将直接用程序实例说明。

程序实例 ch8_6.py：元组切片的应用。

```
1  # ch8_6.py
2  fruits = ('apple', 'orange', 'banana', 'watermelon', 'grape')
3  print(fruits[1:3])
4  print(fruits[:2])
5  print(fruits[1:])
6  print(fruits[-2:])
7  print(fruits[0:5:2])
```

执行结果

```
====================== RESTART: D:/Python/ch8/ch8_6.py ======================
('orange', 'banana')
('apple', 'orange')
('orange', 'banana', 'watermelon', 'grape')
('watermelon', 'grape')
('apple', 'banana', 'grape')
```

8-7 方法与函数

应用在列表上的**方法**或**函数**如果不会更改**元组**内容，则可以将它应用在**元组**上，例如：len( )。如果会更改元组内容，则不可以将它应用在元组上，例如：append( )、insert( ) 或 pop( )。

程序实例 ch8_7.py：列出元组元素长度（个数）。

```
1  # ch8_7.py
2  keys = ('magic', 'xaab', 9099)       # 定义元组元素是字符串与数字
3  print("keys元组长度是 %d " % len(keys))
```

执行结果

```
====================== RESTART: D:\Python\ch8\ch8_7.py ======================
keys元组长度是 3
```

程序实例 ch8_8.py：误用会减少元组元素的方法 pop( )，产生错误的实例。

```
1  # ch8_8.py
2  keys = ('magic', 'xaab', 9099)        # 定义元组元素是字符串与数字
3  key = keys.pop()                      # 错误
```

执行结果

```
====================== RESTART: D:\Python\ch8\ch8_8.py ======================
Traceback (most recent call last):
  File "D:\Python\ch8\ch8_8.py", line 3, in <module>
    key = keys.pop()              # 错误
AttributeError: 'tuple' object has no attribute 'pop'
```

上述指出第 3 行错误是不支持 pop( )，这是因为 pop( ) 将造成元组元素减少。

程序实例 ch8_9.py：误用会增加元组元素的方法 append( )，产生错误的实例。

```
1  # ch8_9.py
2  keys = ('magic', 'xaab', 9099)       # 定义元组元素是字符串与数字
3  keys.append('secret')                # 错误
```

执行结果

```
====================== RESTART: D:\Python\ch8\ch8_9.py ======================
Traceback (most recent call last):
  File "D:\Python\ch8\ch8_9.py", line 3, in <module>
    keys.append('secret')                 # 错误
AttributeError: 'tuple' object has no attribute 'append'
```

8-8 列表与元组数据互换

程序设计过程中，也许会有需要将列表与元组数据类型互换，可以使用下列指令。

list (元组) ：将元组数据类型改为列表。

tuple (列表) ：将列表数据类型改为元组。

程序实例 ch8_10.py：重新设计 ch8_9.py，将元组改为列表的测试。

```
1  # ch8_10.py
2  keys = ('magic', 'xaab', 9099)       # 定义元组元素是字符串与数字
3  list_keys = list(keys)               # 将元组改为列表
4  list_keys.append('secret')           # 增加元素
5  print("打印元组", keys)
6  print("打印列表", list_keys)
```

执行结果

```
====================== RESTART: D:\Python\ch8\ch8_10.py ======================
打印元组 ('magic', 'xaab', 9099)
打印列表 ['magic', 'xaab', 9099, 'secret']
```

上述第 4 行由于 list_keys 已经是列表，所以可以使用 append( ) 方法。

程序实例 ch8_11.py：将列表改为元组的测试。

```
1  # ch8_11.py
2  keys = ['magic', 'xaab', 9099]      # 定义列表元素是字符串与数字
3  tuple_keys = tuple(keys)            # 将列表改为元组
4  print("打印列表", keys)
5  print("打印元组", tuple_keys)
6  tuple_keys.append('secret')         # 增加元素 --- 错误
```

执行结果

```
====================== RESTART: D:\Python\ch8\ch8_11.py ======================
打印列表 ['magic', 'xaab', 9099]
打印元组 ('magic', 'xaab', 9099)
Traceback (most recent call last):
  File "D:\Python\ch8\ch8_11.py", line 6, in <module>
    tuple_keys.append('secret')         # 增加元素 --- 错误
AttributeError: 'tuple' object has no attribute 'append'
```

上述前 5 行程序是正确的，所以可以看到有分别打印列表和元组元素，程序第 6 行的错误是因为 tuple_keys 是元组，不支持使用 append( ) 增加元素。

8-9 其他常用的元组方法

方法	说明
max(tuple)	获得元组内容最大值
min(tuple)	获得元组内容最小值

程序实例 ch8_12.py：元组内建方法 max( )、min( ) 的应用。

```
1  # ch8_12.py
2  tup = (1, 3, 5, 7, 9)
3  print("tup最大值是", max(tup))
4  print("tup最小值是", min(tup))
```

执行结果

```
====================== RESTART: D:/Python/ch8/ch8_12.py ======================
tup最大值是 9
tup最小值是 1
```

8-10 enumerate 对象在元组中的使用

在 6-12 与 7-5 节中都已有说明 enumerate( ) 的用法，有一点当时没有提到，当我们将 enumerate( ) 方法产生的 enumerate 对象转成列表时，其实此列表的配对元素是元组，在此直接以实例解说。

程序实例 ch8_13.py：测试 enumerate 对象转成列表后，原先的元素变成元组数据类型。

```
1  # ch8_13.py
2  drinks = ["coffee", "tea", "wine"]
3  enumerate_drinks = enumerate(drinks)              # 数值初始是0
4  lst = list(enumerate_drinks)
5  print("转成列表输出，初始索引值是 0 = ", lst)
6  print(type(lst[0]))
```

执行结果

```
====================== RESTART: D:\Python\ch8\ch8_13.py ======================
转成列表输出，初始索引值是 0 =  [(0, 'coffee'), (1, 'tea'), (2, 'wine')]
<class 'tuple'>
```

程序实例 8_14.py：将元组转成 enumerate 对象，再转回元组对象。

```
1  # ch8_14.py
2  drinks = ("coffee", "tea", "wine")
3  enumerate_drinks = enumerate(drinks)                # 数值初始是0
4  print("转成元组输出，初始值是 0 = ", tuple(enumerate_drinks))
5
6  enumerate_drinks = enumerate(drinks, start = 10)    # 数值初始是10
7  print("转成元组输出，初始值是10 = ", tuple(enumerate_drinks))
```

执行结果

```
====================== RESTART: D:\Python\ch8\ch8_14.py ======================
转成元组输出，初始值是 0 =  ((0, 'coffee'), (1, 'tea'), (2, 'wine'))
转成元组输出，初始值是10 =  ((10, 'coffee'), (11, 'tea'), (12, 'wine'))
```

程序实例 ch8_15.py：将元组转成 enumerate 对象，再解析这个 enumerate 对象。

```
1  # ch8_15.py
2  drinks = ("coffee", "tea", "wine")
3  # 解析enumerate对象
4  for drink in enumerate(drinks):                # 数值初始是0
5      print(drink)
6  for count, drink in enumerate(drinks):
7      print(count, drink)
8  print("****************")
9  # 解析enumerate对象
10 for drink in enumerate(drinks, 10):            # 数值初始是10
11     print(drink)
12 for count, drink in enumerate(drinks, 10):
13     print(count, drink)
```

执行结果

```
====================== RESTART: D:/Python/ch8/ch8_15.py ======================
(0, 'coffee')
(1, 'tea')
(2, 'wine')
0 coffee
1 tea
2 wine
****************
(10, 'coffee')
(11, 'tea')
(12, 'wine')
10 coffee
11 tea
12 wine
```

8-11 使用 zip( ) 打包多个对象

这是一个内建函数，参数内容主要是两个或更多个**可迭代（iterable）的对象**，如果存在多个对象（例如：列表或元组），可以用 zip( ) 将多个对象打包成 zip 对象，然后未来视需要将此 zip 对象使用 list( ) 转成列表或使用 tuple( ) 转成元组。不过读者要知道，这时对象的元素将是元组。

程序实例 ch8_16.py：zip( ) 的应用。

```
1  # ch8_16.py
2  fields = ['Name', 'Age', 'Hometown']
3  info = ['Peter', '30', 'Chicago']
4  zipData = zip(fields, info)          # 执行zip
5  print(type(zipData))                 # 打印zip数据类型
6  player = list(zipData)               # 将zip数据转成列表
7  print(player)                        # 打印列表
```

执行结果

```
====================== RESTART: D:/Python/ch8/ch8_16.py ======================
<class 'zip'>
[('Name', 'Peter'), ('Age', '30'), ('Hometown', 'Chicago')]
```

如果放在 zip( ) 函数中的**列表**参数长度不相等，由于多出的元素无法匹配，转成列表对象后 zip 对象元素数量将是较短的数量。

程序实例 ch8_17.py：重新设计 ch8_16.py，fields 列表元素数量个数是 3 个，info 列表数量元素个数只有两个，最后 zip 对象元素数量是两个。

```
1  # ch8_17.py
2  fields = ['Name', 'Age', 'Hometown']
3  info = ['Peter', '30']
4  zipData = zip(fields, info)          # 执行zip
5  print(type(zipData))                 # 打印zip数据类型
6  player = list(zipData)               # 将zip数据转成列表
7  print(player)                        # 打印列表
```

执行结果

```
====================== RESTART: D:/Python/ch8/ch8_17.py ======================
<class 'zip'>
[('Name', 'Peter'), ('Age', '30')]
```

如果在 zip( ) 函数内增加“*”符号，相当于可以 unzip( ) 列表。

程序实例 ch8_18.py：扩充设计 ch8_16.py，恢复 zip 前的列表。

```
 1  # ch8_18.py
 2  fields = ['Name', 'Age', 'Hometown']
 3  info = ['Peter', '30', 'Chicago']
 4  zipData = zip(fields, info)          # 执行zip
 5  print(type(zipData))                 # 打印zip数据类型
 6  player = list(zipData)               # 将zip数据转成列表
 7  print(player)                        # 打印列表
 8
 9  f, i = zip(*player)                  # 执行unzip
10  print("fields = ", f)
11  print("info   = ", i)
```

执行结果

```
====================== RESTART: D:/Python/ch8/ch8_18.py ======================
<class 'zip'>
[('Name', 'Peter'), ('Age', '30'), ('Hometown', 'Chicago')]
fields =  ('Name', 'Age', 'Hometown')
info   =  ('Peter', '30', 'Chicago')
```

上述实例中 zip( ) 函数内的参数是列表，其实参数也可以是元组或是混合不同的数据类型，甚至是 3 个或更多个数据。下列是将 zip( ) 应用在 3 个元组的实例。

```
>>> x1 = (1,2,3)
>>> x2 = (4,5,6)
>>> x3 = (7,8,9)
>>> a = zip(x1,x2,x3)
>>> tuple(a)
((1, 4, 7), (2, 5, 8), (3, 6, 9))
```

8-12 生成式

在 7-2-7 节有说明列表生成式，当时的语法是在左右两边用中括号“[”和“]”，读者可能会想是否可以用小括号“(”和“)”，就可以产生**元组生成式**（tuple generator），此时语法如下：

```
num = (n for n in range(6))
```

其实上述并不是产生元组生成式，而是产生**生成式**（generator）对象，这是一个可迭代对象，可以用迭代方式取出内容，也可以用 list( ) 将此生成式变为列表，或是用 tuple( ) 将此生成式变为元组，但是只能使用一次，因为这个生成式对象不会记住所拥有的内容，如果想要第 2 次使用，将得到空列表。

实例 1：建立生成式，同时用迭代输出。

```
>>> x = (n for n in range(3))
>>> type(x)
<class 'generator'>
>>> for n in x:
        print(n)


0
1
2
```

实例 2：建立生成式，同时转成列表，第二次转成元组，结果元组内容是空的。

```
>>> x = (n for n in range(3))
>>> xlst = list(x)
>>> print(xlst)
[0, 1, 2]
>>> xtup = tuple(x)
>>> print(xtup)
()
```

实例 3：建立生成式，同时转成元组，第二次转成列表，结果列表内容是空的。

```
>>> x = (n for n in range(3))
>>> xtup = tuple(x)
>>> print(xtup)
(0, 1, 2)
>>> xlst = list(x)
>>> print(xlst)
[]
```

8-13 制作大型的元组数据

有时想要制作更大型的元组数据结构，例如：元组的元素是列表，可以参考下列实例。

实例：元组的元素是列表。

```
>>> asia = ['Beijing', 'Hongkong', 'Tokyo']
>>> usa = ['Chicago', 'New York', 'Hawaii', 'Los Angeles']
>>> europe = ['Paris', 'London', 'Zurich']
>>> world = asia, usa, europe
>>> type(world)
<class 'tuple'>
>>> world
(['Beijing', 'Hongkong', 'Tokyo'], ['Chicago', 'New York', 'Hawaii', 'Los Angele
s'], ['Paris', 'London', 'Zurich'])
```

8-14 元组的功能

读者也许好奇，元组的数据结构与列表相同，但是元组有不可更改元素内容的限制，为何 Python 要有类似但功能却受限的数据结构存在？原因是元组有下列优点。

1. 可以更安全地保护数据

程序设计中可能会碰上有些数据是永远不会改变的事实，将它存储在元组内，可以安全地被保护。例如，处理图像时对象的长、宽或每一像素的色彩数据，很多都是以元组为数据类型。

2. 增加程序执行速度

元组的结构比列表简单，占用较少的系统资源，程序执行时速度比较快。

当了解了上述元组的优点后，其实未来在设计程序时，如果确定数据可以不更改，就尽量使用元组数据类型吧！

8-15 专题——认识元组 / 统计应用

8-15-1 认识元组

元组由于具有安全、内容不会被更改、数据结构单纯、执行速度快等优点，被大量应用在系统程序设计中，程序设计师喜欢将设计程序所保留的数据以元组形式存储。

在 2-9 节和 3-7-1 节有介绍 divmod( ) 函数，这个函数的返回值是商和余数，当时笔者用下列公式表达了这个函数的用法。

```
商 , 余数 = divmod( 被除数 , 除数 )          # 函数方法
```

更严格地说，divmod( ) 的返回值是元组，所以可以使用元组方式取得商和余数。

程序实例 ch8_19.py：使用元组重新设计 ch3_24.py，计算地球到月球的时间。

```
1  # ch8_19.py
2  dist = 384400                # 地球到月球距离
3  speed = 1225                 # 马赫速度每小时1225千米
4  total_hours = dist // speed  # 计算小时数
5  data = divmod(total_hours, 24)  # 商和余数
6  print("divmod返回的数据类型是 : ", type(data))
7  print("总共需要 %d 天" % data[0])
8  print("%d 小时" % data[1])
```

执行结果

```
===================== RESTART: D:\Python\ch8\ch8_19.py =====================
divmod返回的数据类型是 :  <class 'tuple'>
总共需要 13 天
1 小时
```

从上述第 6 行的执行结果可以看到返回值 data 的数据类型是元组 tuple。若是再看 divmod( ) 函数公式，可以得到第一个参数“商”相当于索引为 0 的元素，第二个参数“余数”相当于索引为 1 的元素。

8-15-2　基础统计应用

假设有一组数据，此数据中有 n 个数据，可以使用下列公式计算它的**平均值**（Mean）、**变异数**（Variance）、**标准偏差**（Standard Deviation，SD，数学符号为 sigma）。

$$\text{平均值：mean} = \frac{\sum_{i=1}^{n} x_i}{n} = \frac{x_1 + x_2 + \cdots + x_n}{n}$$

$$\text{变异数：variance} = \frac{\sum_{i=1}^{n}(x_i - \text{mean})^2}{n-1}$$

$$\text{标准偏差：standarddeviation} = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \text{mean})}{n-1}}$$

由于统计数据将不会更改，所以可以用元组存储处理。如果未来可能调整此数据，则建议使用列表存储处理。下列实例为用元组存储数据。

程序实例 ch8_20.py：计算 5,6,8,9 的平均值、变异数和标准偏差。

```
1  # ch8_20.py
2  # 计算平均值
3  vals = (5,6,8,9)
4  mean = sum(vals) / len(vals)
5  print("平均值 : ", mean)
6
7  # 计算变异数
8  var = 0
9  for v in vals:
10     var += ((v - mean)**2)
11 var = var / (len(vals)-1)
12 print("变异数 : ", var)
13
14 # 计算标准偏差
15 dev = 0
16 for v in vals:
17     dev += ((v - mean)**2)
18 dev = (dev / (len(vals)-1))**0.5
19 print("标准偏差 : ", dev)
```

执行结果

```
====================== RESTART: D:\Python\ch8\ch8_20.py ======================
平均值 :  7.0
变异数 :  3.3333333333333335
标准偏差 :  1.8257418583505538
```

习题

1. 你组织了一个 Python 的读书小组，这个小组成员有 5 个人，John、Peter、Curry、Mike、Kevin，请将这 5 个人的姓名存储在元组内，请使用 for 循环打印这 5 个人的姓名。(8-3 节)

```
======================= RESTART: D:\Python\ex\ex8_1.py =======================
读书会成员
John
Peter
Curry
Mike
Kevin
```

2. 请参考第 1 题，尝试修改 John 为 Johnnason，然后列出所得到的错误消息。(8-4 节)

```
======================= RESTART: D:\Python\ex\ex8_2.py =======================
读书会成员
John
Peter
Curry
Mike
Kevin
Traceback (most recent call last):
  File "D:\Python\ex\ex8_2.py", line 6, in <module>
    bookclub[0] = 'Johnnason'
TypeError: 'tuple' object does not support item assignment
```

3. 请使用重新设置方式，将 5 个小组成员改为 8 人，新增加的 3 人是 Mary、Tom、Carlo，然后打印这 8 个人的姓名。(8-5 节)

```
======================= RESTART: D:\Python\ex\ex8_3.py =======================
原先的读书会成员
John
Peter
Curry
Mike
Kevin
新的读书会成员
John
Peter
Curry
Mike
Kevin
Mary
Tom
Carlo
```

4. 有一个元组的元素有重复 tp = (1,2,3,4,5,2,3,1,4)，请建立一个新元组 newtp，此新元组存储相同但没有重复的元素。提示：需用列表处理，最后转成元组。(8-8 节)

```
======================= RESTART: D:\Python\ex\ex8_4.py =======================
新的元组内容 :  (1, 2, 3, 4, 5)
```

5. season 元组内容是 ('Spring', 'Summer', 'Fall', 'Winter')，chinese 元组内容是 ('春季', '夏季', '秋季', '冬季')，请使用 zip( ) 将这两个元组打包，然后转成列表打印出来。(8-11 节)

```
======================= RESTART: D:/Python/ex/ex8_5.py =======================
[('Spring', '春季'), ('Summer', '夏季'), ('Fall', '秋季'), ('Winter', '冬季')]
```

6. 气象局使用元组记录了北京市过去一周的最高温和最低温度：(8-15 节)

最高温度：30, 28, 29, 31, 33, 35, 32

最低温度：20, 21, 19, 22, 23, 24, 20

请列出过去一周的最高温、最低温和平均温度。

```
======================= RESTART: D:\Python\ex\ex8_6.py =======================
过去一周的最高温度 35
过去一周的最低温度 28
过去一周的平均温度
25.0  24.5  24.0  26.5  28.0  29.5  26.0
```

7. 有一个超商统计一周来入场人数分别是 1100、652、946、821、955、1024、1155。请计算平均值、变异数和标准偏差。(8-15 节)

```
======================= RESTART: D:\Python\ex\ex8_7.py =======================
平均值 :  950.4285714285714
变异数 :  29247.619047619042
标准偏差 :  171.01935284528193
```

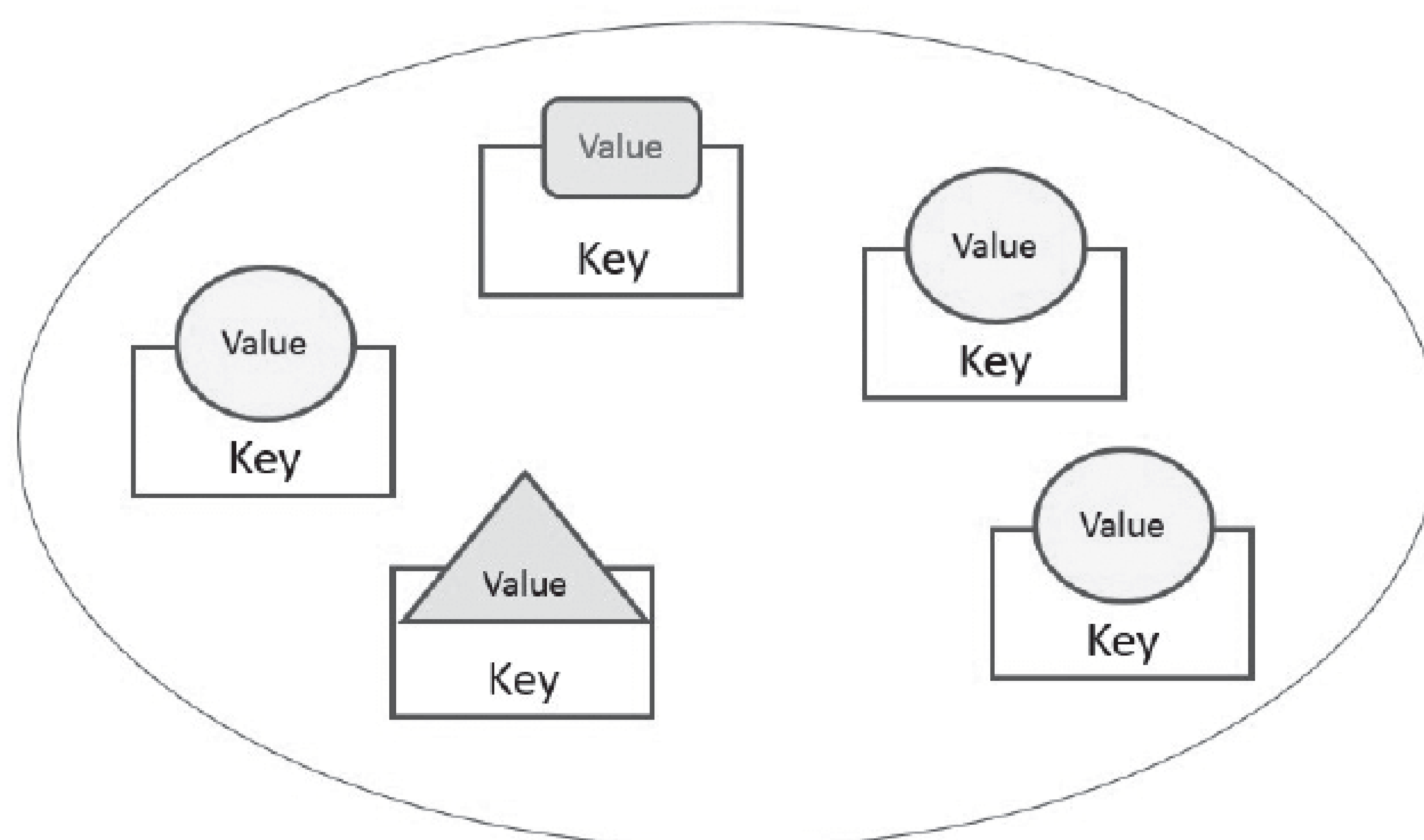
09

第 9 章

字典

本章摘要

列表与元组是依序排列的可称为**序列**数据结构，只要知道元素的特定位置，即可使用**索引**取得元素内容。这一章的重点是介绍**字典**（dict），它并不是依序排列的数据结构，通常可称为**非序列**数据结构，所以无法使用类似列表的索引 [0, 1, …，n] 取得元素内容。



9-1 字典的基本操作

9-1-1 定义字典

字典是一个非序列的数据结构，但是它的元素是用“**键 : 值**”方式**配对存储**，在操作时是用**键**（key）取得**值**（value）的内容，其实在真实的应用中可以将字典数据结构当作正式的字典使用，查询键时，就可以列出相对应的值的内容。本章将穿插各种字典的实例应用。定义字典时，是将“**键 : 值**”放在**大括号**“{ }”内，字典的语法格式如下：

```
name_dict = { 键1: 值1, … , 键n: 值n, }      # name_dict 是字典变量名称
```

字典的键（key）一般常用的是**字符串**或**数字**，在一个字典中不可有重复的键（key）出现。字典的值（value）可以是任何 Python 的数据对象，所以可以是数值、字符串、列表、字典等。最右边的“键 n: 值 n”后的“,”可有可无。

程序实例 ch9_1.py：以水果行和面店为例定义一个字典，同时列出字典。下列字典是设置一斤水果的价格、一碗面的价格，最后使用 type( ) 列出字典数据类型。

```
1  # ch9_1.py
2  fruits = {'西瓜':15, '香蕉':20, '水蜜桃':25}
3  noodles = {'牛肉面':100, '肉丝面':80, '阳春面':60}
4  print(fruits)
5  print(noodles)
6  # 列出字典数据类型
7  print("字典fruits数据类型是: ",type(fruits))
```

执行结果

```
====================== RESTART: D:\Python\ch9\ch9_1.py ======================
{'西瓜': 15, '香蕉': 20, '水蜜桃': 25}
{'牛肉面': 100, '肉丝面': 80, '阳春面': 60}
字典fruits数据类型是:  <class 'dict'>
```

在使用 Python 设计打斗游戏时，玩家通常扮演英雄的角色，敌军可以用字典方式存储，例如，可以用不同颜色的标记设置敌军的小兵，每一个敌军的小兵给予一个分数，这样可以由打死敌军数量再统计游戏得分，可以用下列方式定义字典内容。

程序实例 ch9_2.py：定义 soldier0 字典，tag 和 score 是键，red 和 3 是值。

```
1  # ch9_2.py
2  soldier0 = {'tag':'red', 'score':3}
3  print(soldier0)
```

执行结果

```
===================== RESTART: D:\Python\ch9\ch9_2.py =====================
{'tag': 'red', 'score': 3}
```

上述是定义红色（red）小兵，分数是 3 分，玩家打死红色小兵得 3 分。

9-1-2　列出字典元素的值

字典的元素是“键 : 值”配对设置，如果想要取得元素的值，可以将键当作索引方式处理，因此字典内的元素不可有重复的键，可参考下列实例 ch9_3.py 的第 4 行，例如，下列可返回 fruits 字典水蜜桃键的值。

```
fruits['水蜜桃']                    # 用字典变量['键']取得值
```

程序实例 ch9_3.py：分别列出 ch9_1.py，水果店一斤水蜜桃的价格和面店一碗牛肉面的价格。

```
1  # ch9_3.py
2  fruits = {'西瓜':15, '香蕉':20, '水蜜桃':25}
3  noodles = {'牛肉面':100, '肉丝面':80, '阳春面':60}
4  print("水蜜桃一斤 = ", fruits['水蜜桃'], "元")
5  print("牛肉面一碗 = ", noodles['牛肉面'], "元")
```

执行结果

```
===================== RESTART: D:\Python\ch9\ch9_3.py =====================
水蜜桃一斤 =  25 元
牛肉面一碗 =  100 元
```

程序实例 ch9_4.py：分别列出 ch9_2.py 小兵字典的 tag 和 score 键的值。

```
1  # ch9_4.py
2  soldier0 = {'tag':'red', 'score':3}
3  print("你刚打死标记 %s 小兵" % soldier0['tag'])
4  print("可以得到 ", soldier0['score'], " 分")
```

执行结果

```
===================== RESTART: D:\Python\ch9\ch9_4.py =====================
你刚打死标记 red 小兵
可以得到  3  分
```

有趣地活用“键 : 值”，如果有一字典如下：

fruits = {0:' 西瓜 ', 1:' 香蕉 ', 2:' 水蜜桃 '}

上述字典键是整数时，也可以使用下列方式取得值。

fruit[0]　　　　　　　　# 取得键是 0 的值

程序实例 ch9_4_1.py：列出特定键的值。

```
1  # ch9_4_1.py
2  fruits = {0:'西瓜', 1:'香蕉', 2:'水蜜桃'}
3  print(fruits[0], fruits[1], fruits[2])
```

执行结果

```
==================== RESTART: D:\Python\ch9\ch9_4_1.py ====================
西瓜 香蕉 水蜜桃
```

9-1-3　增加字典元素

可使用下列语法格式增加字典元素：

name_dict[键] = 值　　　　　　# **name_dict 是字典变量**

程序实例 ch9_5.py：为 fruits 字典增加橘子一斤 18 元。

```
1  # ch9_5.py
2  fruits = {'西瓜':15, '香蕉':20, '水蜜桃':25}
3  fruits['橘子'] = 18
4  print(fruits)
5  print("橘子一斤 = ", fruits['橘子'], "元")
```

执行结果

```
====================== RESTART: D:\Python\ch9\ch9_5.py ======================
{'西瓜': 15, '香蕉': 20, '水蜜桃': 25, '橘子': 18}
橘子一斤 =  18 元
```

在设计打斗游戏时，可以使用屏幕坐标标记小兵的位置，下列实例是用 xpos/ypos 标记小兵的 x 坐标 /y 坐标。

程序实例 ch9_6.py：为 soldier0 字典增加 x,y 轴坐标（xpos,ypos）和移动速度（speed）元素，同时列出结果做验证。

```
1  # ch9_6.py
2  soldier0 = {'tag':'red', 'score':3}
3  soldier0['xpos'] = 100
4  soldier0['ypos'] = 30
5  soldier0['speed'] = 'slow'
6  print("小兵的 x 坐标  = ", soldier0['xpos'])
7  print("小兵的 y 坐标  = ", soldier0['ypos'])
8  print("小兵的移动速度 = ", soldier0['speed'])
```

执行结果

```
====================== RESTART: D:\Python\ch9\ch9_6.py ======================
小兵的 x 坐标  =  100
小兵的 y 坐标  =  30
小兵的移动速度 =  slow
```

9-1-4　更改字典元素内容

市面上的水果价格是浮动的，如果发生价格异动可以使用本节的方法更改。

程序实例 ch9_7.py：将 fruits 字典中的一斤香蕉改成 12 元。

```
1  # ch9_7.py
2  fruits = {'西瓜':15, '香蕉':20, '水蜜桃':25}
3  print("旧价格香蕉一斤 = ", fruits['香蕉'], "元")
4  fruits['香蕉'] = 12
5  print("新价格香蕉一斤 = ", fruits['香蕉'], "元")
```

执行结果

```
====================== RESTART: D:\Python\ch9\ch9_7.py ======================
旧价格香蕉一斤 =  20 元
新价格香蕉一斤 =  12 元
```

在设计打斗游戏时，需要时时移动小兵的位置，此时可以使用本节方法时时更改小兵位置。

程序实例 ch9_8.py：依照 soldier 字典 speed 键的值更改小兵位置。

```
1  # ch9_8.py
2  soldier0 = {'tag':'red', 'score':3, 'xpos':100,
3              'ypos':30, 'speed':'slow' }
4  print("小兵的 x,y 旧坐标  = ", soldier0['xpos'], ",", soldier0['ypos'] )
5  if soldier0['speed'] == 'slow':          # 慢
6      x_move = 1
7  elif soldier0['speed'] == 'medium':      # 中
8      x_move = 3
9  else:
10     x_move = 5                           # 快
11 soldier0['xpos'] += x_move
12 print("小兵的 x,y 新坐标  = ", soldier0['xpos'], ",", soldier0['ypos'] )
```

执行结果

```
====================== RESTART: D:\Python\ch9\ch9_8.py ======================
小兵的 x,y 旧坐标  =  100 , 30
小兵的 x,y 新坐标  =  101 , 30
```

上述程序将小兵移动速度分成 3 个等级，slow 是每次 xpos 移动 1 单位（5 和 6 行），medium 是每次 xpos 移动 3 单位（7 和 8 行），另一等级则是每次 xpos 移动 5 单位（9 和 10 行）。第 11 行是执行小兵移动，为了简化条件 y 轴暂不移动。所以可以得到上述小兵 x 轴位置由 100 移到 101。

9-1-5　删除字典特定元素

如果想要删除字典的特定元素，它的语法格式如下：

```
del name_dict[键]                # 可删除特定键的元素
```

程序实例 ch9_9.py：删除 fruits 字典中的西瓜元素。

```
1  # ch9_9.py
2  fruits = {'西瓜':15, '香蕉':20, '水蜜桃':25}
3  print("旧fruits字典内容:", fruits)
4  del fruits['西瓜']
5  print("新fruits字典内容:", fruits)
```

执行结果

```
===================== RESTART: D:\Python\ch9\ch9_9.py =====================
旧fruits字典内容: {'西瓜': 15, '香蕉': 20, '水蜜桃': 25}
新fruits字典内容: {'香蕉': 20, '水蜜桃': 25}
```

9-1-6 字典的 pop( ) 方法

Python 字典的 pop( ) 方法也可以删除字典内特定的元素，同时返回所删除的元素，它的语法格式如下：

```
ret_value = dictObj.pop(key[, default])    # dictObj 是要删除元素的字典
```

上述 key 是要查找删除元素的键，找到时就将该元素从字典内删除，同时将删除键的值返回。当找不到 key 时则返回 default 设置的内容，如果没有设置则导致 KeyError，程序异常终止，在第 15 章将讲解如何处理程序异常终止。

程序实例 ch9_9_1.py：删除字典元素同时可以返回所删除字典元素的应用。

```
1  # ch9_9_1.py
2  fruits = {'西瓜':15, '香蕉':20, '水蜜桃':25}
3  print("旧fruits字典内容:", fruits)
4  objKey = '西瓜'
5  value = fruits.pop(objKey)
6  print("新fruits字典内容:", fruits)
7  print("删除内容:", objKey + ":" + str(value))
```

执行结果

```
==================== RESTART: D:\Python\ch9\ch9_9_1.py ====================
旧fruits字典内容: {'西瓜': 15, '香蕉': 20, '水蜜桃': 25}
新fruits字典内容: {'香蕉': 20, '水蜜桃': 25}
删除内容: 西瓜:15
```

实例 1：所删除的元素不存在，导致“KeyError”，程序异常终止。

```
>>> num = {1:'a',2:'b'}
>>> value = num.pop(3)
Traceback (most recent call last):
  File "<pyshell#229>", line 1, in <module>
    value = num.pop(3)
KeyError: 3
```

实例 2：所删除的元素不存在，打印“does not exist”字符串。

```
>>> num = {1:'a',2:'b'}
>>> value = num.pop(3, 'does no exist')
>>> value
'does no exist'
```

9-1-7 字典的 popitem( ) 方法

Python 字典的 popitem( ) 方法可以随机删除字典内的元素，同时返回所删除的元素，所返回的是元组（key, value），它的语法格式如下：

```
valueTup = dictObj.popitem()          # 可随机删除字典的元素
```

如果字典是空的，会有错误异常产生。

程序实例 ch9_9_2.py：列出所随机删除的字典元素内容。

```
1  # ch9_9_2.py
2  fruits = {'西瓜':15, '香蕉':20, '水蜜桃':25}
3  print("旧fruits字典内容:", fruits)
4  valueTup = fruits.popitem()
5  print("新fruits字典内容:", fruits)
6  print("删除内容:", valueTup)
```

执行结果

```
==================== RESTART: D:\Python\ch9\ch9_9_2.py ====================
旧fruits字典内容: {'西瓜': 15, '香蕉': 20, '水蜜桃': 25}
新fruits字典内容: {'西瓜': 15, '香蕉': 20}
删除内容: ('水蜜桃', 25)
```

9-1-8　删除字典所有元素

Python 提供了 clear( ) 方法将字典的所有元素删除，此时字典仍然存在，不过将变成空的字典。

程序实例 ch9_10.py：使用 clear( ) 方法删除 fruits 字典的所有元素。

```
1  # ch9_10.py
2  fruits = {'西瓜':15, '香蕉':20, '水蜜桃':25}
3  print("旧fruits字典内容:", fruits)
4  fruits.clear()
5  print("新fruits字典内容:", fruits)
```

执行结果

```
==================== RESTART: D:\Python\ch9\ch9_10.py ====================
旧fruits字典内容: {'西瓜': 15, '香蕉': 20, '水蜜桃': 25}
新fruits字典内容: {}
```

9-1-9　删除字典

Python 提供了 del 指令将整个字典删除，字典一经删除就不再存在。它的语法格式如下：

```
del name_dict                         # 删除字典 name_dict
```

程序实例 ch9_11.py：删除字典的测试，这个程序前 4 行是没有任何问题的，第 5 行尝试打印已经被删除的字典，所以产生错误，错误原因是没有定义 fruits 字典。

```
1  # ch9_11.py
2  fruits = {'西瓜':15, '香蕉':20, '水蜜桃':25}
3  print("旧fruits字典内容:", fruits)
4  del fruits
5  print("新fruits字典内容:", fruits)       # 错误!
```

执行结果

```
====================== RESTART: D:\Python\ch9\ch9_11.py ======================
旧fruits字典内容: {'西瓜': 15, '香蕉': 20, '水蜜桃': 25}
Traceback (most recent call last):
  File "D:\Python\ch9\ch9_11.py", line 5, in <module>
    print("新fruits字典内容:", fruits)       # 错误!
NameError: name 'fruits' is not defined
```

9-1-10 建立一个空字典

在程序设计时，也允许先建立一个空字典，建立空字典的语法如下：

```
name_dict = { }                          # name_dict 是字典名称
```

上述字典建立完成后，可以用 9-1-3 节增加字典元素的方式为空字典建立元素。

程序实例 ch9_12.py：建立一个小兵的空字典，然后为小兵建立元素。

```
1  # ch9_12
2  soldier0 = {}             # 建立空字典
3  print("空小兵字典", soldier0)
4  soldier0['tag'] = 'red'
5  soldier0['score'] = 3
6  print("新小兵字典", soldier0)
```

执行结果

```
====================== RESTART: D:\Python\ch9\ch9_12.py ======================
空小兵字典 {}
新小兵字典 {'tag': 'red', 'score': 3}
```

9-1-11 字典的复制

在大型程序开发过程中，为了要保护原字典内容，所以常会需要将字典复制，此时可以使用此方法。

```
new_dict = name_dict.copy( )             # name_dict 会被复制至 new_dict
```

上述所复制的字典是独立存在新地址的字典。

程序实例 ch9_13.py：复制字典，同时列出新字典所在地址，如此可以验证新字典与旧字典是不同的字典。

```
1  # ch9_13.py
2  fruits = {'西瓜':15, '香蕉':20, '水蜜桃':25, '苹果':18}
3  cfruits = fruits.copy()
4  print("地址 = ", id(fruits), "  fruits元素 = ", fruits)
5  print("地址 = ", id(cfruits), "  fruits元素 = ", cfruits)
```

执行结果

```
====================== RESTART: D:\Python\ch9\ch9_13.py ======================
地址 =  47571712   fruits元素 =  {'西瓜': 15, '香蕉': 20, '水蜜桃': 25, '苹果': 18}
地址 =  52212240   fruits元素 =  {'西瓜': 15, '香蕉': 20, '水蜜桃': 25, '苹果': 18}
```

请留意上述说明的是浅拷贝，笔者在 6-8-4 节介绍的**浅拷贝**（copy 或称 shallow copy）与**深拷贝**（deep copy）的概念一样可以应用于字典。如果字典内容包含子对象时，建议使用深拷贝，这样可以更加保护原对象内容。

实例 1：浅拷贝在更改字典子对象内容时，造成原字典子对象内容被修改。

```
>>> a = {'a':[1, 2, 3]}
>>> b = a.copy()
>>> a, b
({'a': [1, 2, 3]}, {'a': [1, 2, 3]})
>>> b['a'].append(4)
>>> a, b
({'a': [1, 2, 3, 4]}, {'a': [1, 2, 3, 4]})
```

上述程序的重点是碰上修改子对象时，原对象内容也被更改了。此外，上述字典内键的值是列表，更多相关知识在 9-4 节会说明。

所以如果要更安全地保护原字典，建议使用深拷贝，

实例 2：深拷贝在更改字典子对象内容时，原字典子对象内容可以不改变。

```
>>> import copy
>>> a = {'a':[1, 2, 3]}
>>> b = copy.deepcopy(a)
>>> a, b
({'a': [1, 2, 3]}, {'a': [1, 2, 3]})
>>> b['a'].append(4)
>>> a, b
({'a': [1, 2, 3]}, {'a': [1, 2, 3, 4]})
```

9-1-12　取得字典元素数量

在列表或元组中使用的方法 len( ) 也可以应用于**字典**，它的语法如下：

```
length = len(name_dict)    # 将传会 name_dict 字典的元素数量给 length
```

程序实例 ch9_14.py：列出空字典和一般字典的元素数量，本程序第 4 行由于是建立空字典，所以第 7 行打印元素数量是 0。

```
1  # ch9_14.py
2  fruits = {'西瓜':15, '香蕉':20, '水蜜桃':25, '苹果':18}
3  noodles = {'牛肉面':100, '肉丝面':80, '阳春面':60}
4  empty_dict = {}
5  print("fruits字典元素数量     = ", len(fruits))
6  print("noodles字典元素数量    = ", len(noodles))
7  print("empty_dict字典元素数量 = ", len(empty_dict))
```

执行结果

```
====================== RESTART: D:\Python\ch9\ch9_14.py ======================
fruits字典元素数量     =  4
noodles字典元素数量    =  3
empty_dict字典元素数量 =  0
```

9-1-13　验证元素是否存在

可以用下列语法验证元素是否存在。

```
键 in name_dict          # 可验证键元素是否存在
```

程序实例 ch9_15.py：这个程序会要求输入“键 : 值”，然后判断此元素是否在 fruits 字典中，如果不在此字典，则将此“键 : 值”加入字典。

```
1  # ch9_15.py
2  fruits = {'西瓜':15, '香蕉':20, '水蜜桃':25}
3  key = input("请输入键(key) = ")
4  value = input("请输入值(value) = ")
5  if key in fruits:
6      print("%s已经在字典了" % key)
7  else:
8      fruits[key] = value
9      print("新的fruits字典内容 = ", fruits)
```

执行结果

```
====================== RESTART: D:\Python\ch9\ch9_15.py ======================
请输入键(key) = 西瓜
请输入值(value) = 15
西瓜已经在字典了
>>>
====================== RESTART: D:\Python\ch9\ch9_15.py ======================
请输入键(key) = 苹果
请输入值(value) = 18
新的fruits字典内容 =  {'西瓜': 15, '香蕉': 20, '水蜜桃': 25, '苹果': '18'}
```

9-1-14 设计字典的可读性技巧

设计大型程序时，字典的元素内容很可能是由长字符串所组成的，碰上这类情况建议从新的一行开始安置每一个元素，如此可以大大增加字典内容的可读性。例如，有一个 players 字典，元素是由“键（球员名字）: 值（球队名称）”所组成。如果使用传统方式设计，将让整个字典定义变得很复杂，如下所示：

```
players = {'Stephen Curry':'Golden State Warriors','Kevin Durant':'Golden State Warriors',
'Lebron James':'Cleveland Cavaliers', 'Paul Gasol':'San Antonio Spurs'}
```

碰上这类字典，建议一行定义一个元素，如下所示。

```
players = {'Stephen Curry':'Golden State Warriors',
           'Kevin Durant':'Golden State Warriors',
           'Lebron James':'Cleveland Cavaliers',
           'Paul Gasol':'San Antonio Spurs'}
```

程序实例 ch9_16.py：字典元素是长字符串的应用。

```
1  # ch9_16.py
2  players = {'Stephen Curry':'Golden State Warriors',
3             'Kevin Durant':'Golden State Warriors',
4             'Lebron James':'Cleveland Cavaliers',
5             'Paul Gasol':'San Antonio Spurs',
6            }
7  print("Stephen Curry是 %s 的球员" % players['Stephen Curry'])
8  print("Kevin Durant是 %s 的球员" % players['Kevin Durant'])
9  print("Paul Gasol是 %s 的球员" % players['Paul Gasol'])
```

执行结果

```
====================== RESTART: D:\Python\ch9\ch9_16.py ======================
Stephen Curry是 Golden State Warriors 的球员
Kevin Durant是 Golden State Warriors 的球员
Paul Gasol是 San Antonio Spurs 的球员
```

9-1-15　合并字典 update( )

如果想要将两个字典合并，可以使用 update( ) 方法。

程序实例 ch9_16_1.py：字典合并的应用，经销商 A（dealerA）销售 Nissan、Toyota 和 Lexus 这 3 个品牌的车子，经销商 B（dealerB）销售 BMW、Benz 这两个品牌的车子，设计程序当经销商 A 并购了经销商 B 后，列出经销商 A 所销售的车子。

```
1  # ch9_16_1.py
2  dealerA = {1:'Nissan', 2:'Toyota', 3:'Lexus'}
3  dealerB = {11:'BMW', 12:'Benz'}
4  dealerA.update(dealerB)
5  print(dealerA)
```

执行结果

```
============= RESTART: D:/Python/ch9/ch9_16_1.py =============
san', 2: 'Toyota', 3: 'Lexus', 11: 'BMW', 12: 'Benz'}
```

在合并字典时，特别需要注意的是，如果发生键（key）相同则第 2 个字典的值可以取代原先字典的值，所以设计字典合并时要特别注意。

程序实例 ch9_16_2.py：重新设计 ch9_16_1.py，经销商 A 和经销商 B 所销售的汽车品牌发生键相同，造成经销商 A 并购经销商 B 时，原先经销商 A 销售的汽车品牌被覆盖，这个程序中原先经销商 A 销售的 Lexus 品牌将被覆盖。

```
1  # ch9_16_2.py
2  dealerA = {1:'Nissan', 2:'Toyota', 3:'Lexus'}
3  dealerB = {3:'BMW', 4:'Benz'}
4  dealerA.update(dealerB)
5  print(dealerA)
```

执行结果

```
==================== RESTART: D:\Python\ch9\ch9_16_2.py ====================
{1: 'Nissan', 2: 'Toyota', 3: 'BMW', 4: 'Benz'}
```

9-1-16　dict( )

在数据处理中可能会碰上双值序列的数据，如下所示：

[['日本', '东京'], ['泰国', '曼谷'], ['英国', '伦敦']]

上述是普通的键 / 值序列，可以使用 dict( ) 将此序列转成字典，其中，双值序列的第一个是键，第二个是值。

程序实例 ch9_16_3.py：将双值序列的列表转成字典。

```
1  # ch9_16_3.py
2  nation = [['日本','东京'],['泰国','曼谷'],['英国','伦敦']]
3  nationDict = dict(nation)
4  print(nationDict)
```

执行结果

```
==================== RESTART: D:\Python\ch9\ch9_16_3.py ====================
{'日本': '东京', '泰国': '曼谷', '英国': '伦敦'}
```

如果上述元素是元组，例如（'日本','东京'）也可以完成相同的工作。

实例 1：将将双值序列的列表转成字典，其中元素是元组。

```
>>> x = [('a','b'), ('c','d')]
>>> y = dict(x)
>>> y
{'a': 'b', 'c': 'd'}
```

实例 2：下列是双值序列是元组的其他实例。

```
>>> x = ('ab', 'cd', 'ed')
>>> y = dict(x)
>>> y
{'a': 'b', 'c': 'd', 'e': 'd'}
```

9-1-17　再谈 zip( )

在 8-11 节已经说明 zip( ) 的用法，其实也可以使用 zip( ) 快速建立字典。

实例 1：zip( ) 应用 1。

```
>>> mydict = dict(zip('abcde', range(5)))
>>> print(mydict)
{'a': 0, 'b': 1, 'c': 2, 'd': 3, 'e': 4}
```

实例 2：zip( ) 应用 2。

```
>>> mydict = dict(zip(['a', 'b', 'c'], range(3)))
>>> print(mydict)
{'a': 0, 'b': 1, 'c': 2}
```

9-1-18　人工智能——语意分析

人工智能应用于海量的信息处理、分析，称为**语意分析**，例如，分析发掘网友每天在微信或脸书发表文章的潜在主题。这个分析过程其实是将每篇文章做分析，分析方式是将文章内容切割成字典模式，以字典方式存储。例如，有一篇文章“韩冰喜欢吃香蕉，也喜欢吃菠萝”，可以处理成下列字典：

{"韩冰":1, "喜欢":2, "吃":2, "香蕉":1, "也":1, "菠萝":1}

从上述字典已经可以筛选文章的基本主题了，至于更进一步的分析读者可以参考有关人工智能语意分析的书籍。

9-2　遍历字典

大型程序设计中，字典用久了会产生相当数量的元素，也许是几千笔或几十万笔或更多。本节将说明如何遍历字典的键、值、键 : 值对。

9-2-1　遍历字典的键：值

Python 有提供方法 items( )，可以让我们取得字典 **"键 : 值"** 配对的元素，若是以 ch9_16.py 的 players 字典为实例，可以使用 for 循环加上 items( ) 方法，如下所示。


```
for name, team in players.items():
    print("\n姓名: ", name)
    print("队名: ", team)
```

上述只要尚未完成遍历字典，for 循环将持续进行，如此就可以完成遍历字典，同时返回所有的 **"键 : 值"**。

程序实例 ch9_17.py：列出 players 字典的所有元素，相当于所有球员数据。

```
1  # ch9_17.py
2  players = {'Stephen Curry':'Golden State Warriors',
3             'Kevin Durant':'Golden State Warriors',
4             'Lebron James':'Cleveland Cavaliers',
5             'Paul Gasol':'San Antonio Spurs'}
6  for name, team in players.items():
7      print("\n姓名: ", name)
8      print("队名: ", team)
```

执行结果

```
====================== RESTART: D:\Python\ch9\ch9_17.py ======================

姓名:  Stephen Curry
队名:  Golden State Warriors

姓名:  Kevin Durant
队名:  Golden State Warriors

姓名:  Lebron James
队名:  Cleveland Cavaliers

姓名:  Paul Gasol
队名:  San Antonio Spurs
```

上述实例的执行结果中虽然元素出现顺序与程序第 2 行到第 5 行的顺序相同，不过读者需了解在 Python 的直译器中并不保证未来一定会保持相同顺序，因为字典是一个**无序**的数据结构，Python 只会保持"键 : 值"而不会关注元素的排列顺序。

读者需留意 items( ) 方法所返回的其实是一个元组，我们只是使用 name, team 分别取得所返回的元组内容，可参考下列实例。

```
>>> d = {1:'a', 2:'b'}
>>> for x in d.items():
        print(type(x))
        print(x)


<class 'tuple'>
(1, 'a')
<class 'tuple'>
(2, 'b')
```

9-2-2 遍历字典的键

有时候我们不想要取得字典的值（value），只想要键（keys），Python 有提供方法 keys( )，让我们取得字典的键的内容，若是以 ch9_16.py 的 players 字典为实例，可以使用 for 循环加上 keys( ) 方法，如下所示：

```
for name in players.keys( ):
    print("姓名: ", name)
```

上述 for 循环会依次将 players 字典的键返回。

程序实例 ch9_18.py：列出 players 字典所有的键（keys），此例是所有球员名字。

```
1  # ch9_18.py
2  players = {'Stephen Curry':'Golden State Warriors',
3             'Kevin Durant':'Golden State Warriors',
4             'Lebron James':'Cleveland Cavaliers',
5             'Paul Gasol':'San Antonio Spurs'}
6  for name in players.keys( ):
7      print("姓名: ", name)
```

执行结果

```
===================== RESTART: D:\Python\ch9\ch9_18.py =====================
姓名:  Stephen Curry
姓名:  Kevin Durant
姓名:  Lebron James
姓名:  Paul Gasol
```

其实上述实例第 6 行也可以省略 keys( ) 方法，而获得一样的结果，未来读者设计程序时是否使用 keys( )，可自行决定，细节可参考 ch9_19.py 的第 6 行。

程序实例 ch9_19.py：重新设计 ch9_18.py，此程序省略了 keys( ) 方法，但增加了一些输出问候语句。

```
1  # ch9_19.py
2  players = {'Stephen Curry':'Golden State Warriors',
3             'Kevin Durant':'Golden State Warriors',
4             'Lebron James':'Cleveland Cavaliers',
5             'Paul Gasol':'San Antonio Spurs'}
6  for name in players:
7      print(name)
8      print("Hi! %s 我喜欢看你在 %s 的表现" % (name, players[name]))
```

执行结果

```
===================== RESTART: D:\Python\ch9\ch9_19.py =====================
Stephen Curry
Hi! Stephen Curry 我喜欢看你在 Golden State Warriors 的表现
Kevin Durant
Hi! Kevin Durant 我喜欢看你在 Golden State Warriors 的表现
Lebron James
Hi! Lebron James 我喜欢看你在 Cleveland Cavaliers 的表现
Paul Gasol
Hi! Paul Gasol 我喜欢看你在 San Antonio Spurs 的表现
```

9-2-3　依键排序与遍历字典

Python 的字典功能并不会处理排序，如果想要遍历字典同时列出排序结果，可以使用方法 sorted( )。

程序实例 ch9_20.py：重新设计程序实例 ch9_19.py，但是名字将以排序方式列出结果，这个程序的重点是第 6 行。

```
1  # ch9_20.py
2  players = {'Stephen Curry':'Golden State Warriors',
3             'Kevin Durant':'Golden State Warriors',
4             'Lebron James':'Cleveland Cavaliers',
5             'Paul Gasol':'San Antonio Spurs'}
6  for name in sorted(players.keys( )):
7      print(name)
8      print("Hi! %s 我喜欢看你在 %s 的表现" % (name, players[name]))
```

执行结果

```
====================== RESTART: D:\Python\ch9\ch9_20.py ======================
Kevin Durant
Hi! Kevin Durant 我喜欢看你在 Golden State Warriors 的表现
Lebron James
Hi! Lebron James 我喜欢看你在 Cleveland Cavaliers 的表现
Paul Gasol
Hi! Paul Gasol 我喜欢看你在 San Antonio Spurs 的表现
Stephen Curry
Hi! Stephen Curry 我喜欢看你在 Golden State Warriors 的表现
```

9-2-4　遍历字典的值

Python 有提供方法 values( )，可以取得字典值的列表，若是以 ch9_16.py 的 players 字典为实例，可以使用 for 循环加上 values( ) 方法，如下所示。

程序实例 ch9_21.py：列出 players 字典的值列表。

```
1  # ch9_21.py
2  players = {'Stephen Curry':'Golden State Warriors',
3             'Kevin Durant':'Golden State Warriors',
4             'Lebron James':'Cleveland Cavaliers',
5             'Paul Gasol':'San Antonio Spurs'}
6  for team in players.values( ):
7      print(team)
```

执行结果

```
====================== RESTART: D:\Python\ch9\ch9_21.py ======================
Golden State Warriors
Golden State Warriors
Cleveland Cavaliers
San Antonio Spurs
```

上述 Golden State Warriors 重复出现，在字典的应用中**键**不可以有重复，**值**可以重复，如果希望所列出的值不要重复，可以使用集合（set）的概念使用 set( ) 函数，例如，将第 7 行改为下列所示即可，这个实例放在本书代码文件 ch9_21_1.py 中，读者可自行参考。这是第 10 章的主题，更多

细节将在第 10 章解说。

```
6    for team in set(players.values()):
```

下列是执行结果，可以发现 Golden State Warriors 不重复了。

```
==================== RESTART: D:\Python\ch9\ch9_21_1.py ====================
Cleveland Cavaliers
San Antonio Spurs
Golden State Warriors
```

9-2-5　依值排序与遍历字典的值

如果有一个 oldDict 字典想要依字典的值（value）排序，可以使用下列函数方法，这时会返回新的排序结果列表：

```
newList = sorted(oldDict.items( ), key=lambda item:item[1])
```

此列表 newList 的元素是**元组**，元组内有两个元素分别是原先字典的**键**和**值**。

程序实例 ch9_21_2.py：将 noodles 字典依键的值排序，此例是依面的售价由小到大排序，转成列表，同时打印。

```
1  # ch9_21_2.py
2  noodles = {'牛肉面':100, '肉丝面':80, '阳春面':60,
3             '打卤面':90, '麻酱面':70}
4  print(noodles)
5  noodlesList = sorted(noodles.items(), key=lambda item:item[1])
6  print(noodlesLst)
```

执行结果

```
==================== RESTART: D:\Python\ch9\ch9_21_2.py ====================
{'牛肉面': 100, '肉丝面': 80, '阳春面': 60, '大卤面': 90, '麻酱面': 70}
[('阳春面', 60), ('麻酱面', 70), ('肉丝面', 80), ('打卤面', 90), ('牛肉面', 100)]
```

从上述执行结果可以看到 noodlesList 是一个列表，列表元素是元组，每个元组有两个元素，列表内容已经依面的售价由低到高排列。如果想要继续扩充列出最便宜的面或是最贵的面，可以使用下列函数。

```
max(noodles.values())        # 最贵的面
min(noodles.values())        # 最便宜的面
```

9-3　建立字典列表

读者可以思考一下程序实例 ch9_2.py，我们建立了小兵 soldier0 字典，在真实的游戏设计中为了让玩家展现雄风，玩家将面对数十、数百或更多个小兵所组成的敌军，为了管理这些小兵，可以将每个小兵当作一个字典，字典内则有小兵的各种信息，然后将这些小兵字典放入**列表**（list）内。

程序实例 ch9_22.py：建立 3 个小兵字典，然后将小兵组成列表。

```
1  # ch9_22.py
2  soldier0 = {'tag':'red', 'score':3, 'speed':'slow'}          # 建立小兵
3  soldier1 = {'tag':'blue', 'score':5, 'speed':'medium'}
4  soldier2 = {'tag':'green', 'score':10, 'speed':'fast'}
5  armys = [soldier0, soldier1, soldier2]                        # 小兵组成列表
6  for army in armys:                                            # 打印小兵
7      print(army)
```

执行结果

```
====================== RESTART: D:\Python\ch9\ch9_22.py ======================
{'tag': 'red', 'score': 3, 'speed': 'slow'}
{'tag': 'blue', 'score': 5, 'speed': 'medium'}
{'tag': 'green', 'score': 10, 'speed': 'fast'}
```

程序设计中如果每个小兵都要个别设计这样太没效率了，可以使用 7-2 节的 range( ) 函数处理这类问题。

程序实例 ch9_23.py：使用 range( ) 建立 50 个小兵，tag 是 red、score 是 3、speed 是 slow。

```
1   # ch9_23.py
2   armys = []                          # 建立小兵空列表
3   # 建立50个小兵
4   for soldier_number in range(50):
5       soldier = {'tag':'red', 'score':3, 'speed':'slow'}
6       armys.append(soldier)
7   # 打印前3个小兵
8   for soldier in armys[:3]:
9       print(soldier)
10  # 打印小兵数量
11  print("小兵数量 = ", len(armys))
```

执行结果

```
====================== RESTART: D:\Python\ch9\ch9_23.py ======================
{'tag': 'red', 'score': 3, 'speed': 'slow'}
{'tag': 'red', 'score': 3, 'speed': 'slow'}
{'tag': 'red', 'score': 3, 'speed': 'slow'}
小兵数量 =  50
```

读者可能会想上述小兵各种特征都相同，用处可能不大，其实对 Python 而言，虽然 50 个特征相同的小兵放在列表内，其实每个小兵都是独立的，可用索引方式存取。通常可以在游戏过程中使用 if 语句和 for 循环处理。

程序实例 ch9_24.py：重新设计 ch9_23.py 建立 50 个小兵，但是将编号第 36 ~ 38 名的小兵改成 tag 是 blue、score 是 5、speed 是 medium。

```
1   # ch9_24.py
2   armys = []                          # 建立小兵空列表
3   # 建立50个小兵
4   for soldier_number in range(50):
5       soldier = {'tag':'red', 'score':3, 'speed':'slow'}
6       armys.append(soldier)
7   # 打印前3个小兵
8   print("前3名小兵资料")
9   for soldier in armys[:3]:
10      print(soldier)
11  # 更改编号36 ~ 38的小兵
12  for soldier in armys[35:38]:
13      if soldier['tag'] == 'red':
14          soldier['tag'] = 'blue'
15          soldier['score'] = 5
16          soldier['speed'] = 'medium'
17  # 打印编号35 ~ 40的小兵
18  print("打印编号35 ~ 40小兵数据")
19  for soldier in armys[34:40]:
20      print(soldier)
```

执行结果

```
======================= RESTART: D:\Python\ch9\ch9_24.py =======================
前3名小兵资料
{'tag': 'red', 'score': 3, 'speed': 'slow'}
{'tag': 'red', 'score': 3, 'speed': 'slow'}
{'tag': 'red', 'score': 3, 'speed': 'slow'}
打印编号35～40小兵数据
{'tag': 'red', 'score': 3, 'speed': 'slow'}
{'tag': 'blue', 'score': 5, 'speed': 'medium'}
{'tag': 'blue', 'score': 5, 'speed': 'medium'}
{'tag': 'blue', 'score': 5, 'speed': 'medium'}
{'tag': 'red', 'score': 3, 'speed': 'slow'}
{'tag': 'red', 'score': 3, 'speed': 'slow'}
```

当然读者可以使用相同方式扩充上述实例，这个将当作习题给读者练习。

9-4 字典内键的值是列表

在 Python 的应用中也允许将列表放在字典内，这时**列表**将是字典某键的**值**。如果想要遍历这类数据结构，需要使用嵌套循环和字典的方法 items( )，**外层循环**是取得字典的**键**，**内层循环**则是将含列表的**值**拆解。下面是定义 sports 字典的实例。

```
3  sports = {'Curry':['篮球', '美式足球'],
4             'Durant':['棒球'],
5             'James':['美式足球', '棒球', '篮球']}
```

上述 sports 字典内含 3 个"**键**：**值**"配对元素，其中，**值**的部分都是列表。程序设计时外层循环配合 items( ) 方法，设计如下。

```
7  for name, favorite_sport in sports.items():
8          print("%s 喜欢的运动是: " % name)
```

上述设计后，**键**内容会传给 name 变量，**值**内容会传给 favorite_sport 变量，所以第 8 行将可打印**键**内容。内层循环主要是将 favorite_sport 列表内容拆解，它的设计如下。

```
10          for sport in favorite_sport:
11              print("    ", sport)
```

上述列表内容会随循环传给 sport 变量，所以第 11 行可以列出结果。

程序实例 ch9_25.py：字典内含列表元素的应用，本程序会先定义内含字符串的字典，然后再拆解打印。

```
1  # ch9_25.py
2  # 建立内含字符串的字典
3  sports = {'Curry':['篮球', '美式足球'],
4             'Durant':['棒球'],
5             'James':['美式足球', '棒球', '篮球']}
6  # 打印key名字 + 字符串'喜欢的运动'
7  for name, favorite_sport in sports.items():
8          print("%s 喜欢的运动是: " % name)
9  # 打印value,这是列表
10         for sport in favorite_sport:
11             print("    ", sport)
```

执行结果

```
====================== RESTART: D:\Python\ch9\ch9_25.py ======================
Curry 喜欢的运动是:
    篮球
    美式足球
Durant 喜欢的运动是:
    棒球
James 喜欢的运动是:
    美式足球
    棒球
    篮球
```

9-5 字典内键的值是字典

在 Python 的应用中也允许将**字典**放在**字典**内，这时**字典**将是字典某键的**值**。假设微信（wechat_account）账号是用字典存储，**键**有两个**值**是由**另外的字典**组成，这个内部字典另有 3 个键，分别是 last_name、first_name 和 city，下列是设计实例。

```
3  wechat_account = {'cshung':{
4                          'last_name':'洪',
5                          'first_name':'锦魁',
6                          'city':'台北'},
7                    'kevin':{
8                          'last_name':'郑',
9                          'first_name':'义盟',
10                         'city':'北京'}}
```

至于打印方式同样需使用 items( ) 函数，可参考下列实例。

程序实例 ch9_26.py：列出字典内含字典的内容。

```
1  # ch9_26.py
2  # 建立内含字典的字典
3  wechat_account = {'cshung':{
4                          'last_name':'洪',
5                          'first_name':'锦魁',
6                          'city':'台北'},
7                    'kevin':{
8                          'last_name':'郑',
9                          'first_name':'义盟',
10                         'city':'北京'}}
11 # 打印内含字典的字典
12 for account, account_info in wechat_account.items( ):
13     print("使用者账号 = ", account)                    # 打印键(key)
14     name = account_info['last_name'] + " " + account_info['first_name']
15     print("姓名       = ", name)                       # 打印值(value)
16     print("城市       = ", account_info['city'])       # 打印值(value)
```

执行结果

```
====================== RESTART: D:\Python\ch9\ch9_26.py ======================
使用者账号 =  cshung
姓名       =  洪 锦魁
城市       =  台北
使用者账号 =  kevin
姓名       =  郑 义盟
城市       =  北京
```

9-6 while 循环在字典中的应用

这一节的内容主要是将 while 循环应用在字典上。

程序实例 ch9_27.py：这是一个市场梦幻旅游地点调查的实例，此程序会要求输入名字以及梦幻旅游地点，然后存入 survey_dict 字典，其中，键是 name，值是 travel_location。输入完后程序会询问是否有人要输入，y 表示有，n 表示没有则程序结束，程序结束前会输出市场调查结果。

```
1  # ch9_27.py
2  survey_dict = {}                        # 建立市场调查空字典
3  market_survey = True                    # 设置循环布尔值
4
5  # 读取参加市场调查者姓名和梦幻旅游景点
6  while market_survey:
7      name = input("\n请输入姓名  : ")
8      travel_location = input("梦幻旅游景点: ")
9
10 # 将输入存入survey_dict字典
11     survey_dict[name] = travel_location
12
13 # 可由此决定是否离开市场调查
14     repeat = input("是否有人要参加市场调查?(y/n) ")
15     if repeat != 'y':                   # 不是输入y,则离开while循环
16         market_survey = False
17
18 # 市场调查结束
19 print("\n\n以下是市场调查的结果")
20 for user, location in survey_dict.items( ):
21     print(user, "梦幻旅游景点: ", location)
```

执行结果

```
====================== RESTART: D:\Python\ch9\ch9_27.py ======================

请输入姓名  : Peter
梦幻旅游景点: Beijing
是否有人要参加市场调查?(y/n) y

请输入姓名  : Kevin
梦幻旅游景点: Hong Kong
是否有人要参加市场调查?(y/n) n


以下是市场调查的结果
Peter 梦幻旅游景点:  Beijing
Kevin 梦幻旅游景点:  Hong Kong
```

有时候设计一个较长的程序时，若是适度空行则整个程序的可读性会比较好，上述程序中分别在第 9、12 和 17 行空一行的目的就是如此。

9-7 字典常用的函数和方法

9-7-1 len( )

可以列出字典元素的个数。

程序实例 ch9_28py：列出字典以及字典内的字典元素的个数。

```
1  # ch9_28.py
2  # 建立内含字典的字典
3  wechat_account = {'cshung':{
4                              'last_name':'洪',
5                              'first_name':'锦魁',
6                              'city':'台北'},
7                        'kevin':{
8                              'last_name':'郑',
9                              'first_name':'义盟',
10                             'city':'北京'}}
11 # 打印字典元素个数
12 print("wechat_account字典元素个数       ", len(wechat_account))
13 print("wechat_account['cshung']元素个数 ", len(wechat_account['cshung']))
14 print("wechat_account['kevin']元素个数  ", len(wechat_account['kevin']))
```

执行结果

```
====================== RESTART: D:\Python\ch9\ch9_28.py ======================
wechat_account字典元素个数         2
wechat_account['cshung']元素个数   3
wechat_account['kevin']元素个数    3
```

9-7-2　fromkeys( )

这是建立字典的一个方法，它的语法格式如下：

```
name_dict = dict.fromkeys(seq[, value])            # 使用 seq 序列建立字典
```

上述语句会使用 seq 序列建立字典，序列内容将是字典的键，如果没有设置 value 则用 none 当字典键的值。

程序实例 ch9_29.py：分别使用列表和元组建立字典。

```
1  # ch9_29.py
2  # 将列表转成字典
3  seq1 = ['name', 'city']          # 定义列表
4  list_dict1 = dict.fromkeys(seq1)
5  print("字典1 ", list_dict1)
6  list_dict2 = dict.fromkeys(seq1, 'Chicago')
7  print("字典2 ", list_dict2)
8  # 将元组转成字典
9  seq2 = ('name', 'city')          # 定义元组
10 tup_dict1 = dict.fromkeys(seq2)
11 print("字典3 ", tup_dict1)
12 tup_dict2 = dict.fromkeys(seq2, 'New York')
13 print("字典4 ", tup_dict2)
```

执行结果

```
====================== RESTART: D:\Python\ch9\ch9_29.py ======================
字典1  {'name': None, 'city': None}
字典2  {'name': 'Chicago', 'city': 'Chicago'}
字典3  {'name': None, 'city': None}
字典4  {'name': 'New York', 'city': 'New York'}
```

9-7-3 get( )

查找字典的键，如果键存在则返回该键的值，如果不存在则返回默认值。

ret_value = dict.get(key[, default=none])　　# dict 是要查找的字典

key 是要查找的键，如果找不到 key 则返回 default 的值（如果没设 default 值就返回 none）。

程序实例 ch9_30.py：get( ) 方法的应用。

```
1  # ch9_30.py
2  fruits = {'Apple':20, 'Orange':25}
3  ret_value1 = fruits.get('Orange')
4  print("Value = ", ret_value1)
5  ret_value2 = fruits.get('Grape')
6  print("Value = ", ret_value2)
7  ret_value3 = fruits.get('Grape', 10)
8  print("Value = ", ret_value3)
```

执行结果

```
====================== RESTART: D:\Python\ch9\ch9_30.py ======================
Value =  25
Value =  None
Value =  10
```

9-7-4 setdefault( )

这个方法基本上与 get( ) 相同，不同之处在于 get( ) 方法不会改变字典内容。使用 setdefault( ) 方法时若所查找的键不存在，会将“**键 : 值**”加入字典，如果有设置默认值则将**键 : 默认值**加入字典，如果没有设置默认值则将键 :none 加入字典。

ret_value = dict.setdefault(key[, default=none])　# dict 是要查找的字典

程序实例 ch9_30_1.py：setdefault( ) 方法，键在字典内的应用。

```
1  # ch9_30_1.py
2  # key在字典内
3  fruits = {'Apple':20, 'Orange':25}
4  ret_value = fruits.setdefault('Orange')
5  print("Value = ", ret_value)
6  print("fruits字典", fruits)
7  ret_value = fruits.setdefault('Orange',100)
8  print("Value = ", ret_value)
9  print("fruits字典", fruits)
```

执行结果

```
===================== RESTART: D:\Python\ch9\ch9_30_1.py =====================
Value =  25
fruits字典 {'Apple': 20, 'Orange': 25}
Value =  25
fruits字典 {'Apple': 20, 'Orange': 25}
```

程序实例 ch9_30_2.py：setdefault( ) 方法，键不在字典内的应用。

```
1  # ch9_30_2.py
2  person = {'name':'John'}
3  print("原先字典内容", person)
4
5  # 'age'键不存在
6  age = person.setdefault('age')
7  print("增加age键 ", person)
8  print("age = ", age)
9
10 # 'sex'键不存在
11 sex = person.setdefault('sex', 'Male')
12 print("增加sex键 ", person)
13 print("sex = ", sex)
```

执行结果

```
==================== RESTART: D:\Python\ch9\ch9_30_2.py ====================
原先字典内容 {'name': 'John'}
增加age键  {'name': 'John', 'age': None}
age =  None
增加sex键  {'name': 'John', 'age': None, 'sex': 'Male'}
sex =  Male
```

9-8 制作大型的字典数据

有时想要制作更大型的字典数据结构，例如，字典的键是地球的洲名，键的值是该洲几个城市名称，可以参考下列实例。

实例 1：字典的元素的值是列表。

```
>>> asia = ['Beijing', 'Hongkong', 'Tokyo']
>>> usa = ['Chicago', 'New York', 'Hawaii', 'Los Angeles']
>>> europe = ['Paris', 'London', 'Zurich']
>>> world = {'Asia':asia, 'Usa':usa, 'Europe':europe}
>>> type(world)
<class 'dict'>
>>> world
{'Asia': ['Beijing', 'Hongkong', 'Tokyo'], 'Usa': ['Chicago', 'New York', 'Hawai
i', 'Los Angeles'], 'Europe': ['Paris', 'London', 'Zurich']}
```

在设计大型程序时，必须记住字典的键是不可变的，所以不可以将列表、字典或是第 10 章将介绍的集合当作字典的键，不过可以将元组当作字典的键。例如，在 4-7-4 节可以知道地球上每个位置是用（纬度 , 经度）当作标记，所以可以使用经纬度当作字典的键。

实例 2：使用经纬度当作字典的键，值是地点名称。

```
>>> loc = {
        (25.0542, 121.5168):'台北车站',
        (22.2838, 114.1731):'红磡车站'
        }

>>> type(loc)

<class 'dict'>
>>> loc

{(25.0542, 121.5168): '台北车站', (22.2838, 114.1731): '红磡车站'}
```

9-9 专题——文件分析 / 字典生成式 / 英汉字典 / 文件加密

9-9-1 传统方式分析文章的文字与字数

程序实例 ch9_31.py：这个项目主要是设计一个程序，可以记录一段英文文字，或是一篇文章所有单字以及每个单字的出现次数，这个程序会用单字当作字典的键（key），用值（value）当作该单字出现的次数。

```
1  # ch9_31.py
2  song = """Are you sleeping, are you sleeping, Brother John, Brother John?
3  Morning bells are ringing, morning bells are ringing.
4  Ding ding dong, Ding ding dong."""
5  mydict = {}                          # 空字典未来存储单字计数结果
6  print("原始歌曲")
7  print(song)
8
9  # 以下是将歌曲大写字母全部改成小写
10 songLower = song.lower()             # 歌曲改为小写
11 print("小写歌曲")
12 print(songLower)
13
14 # 将歌曲的标点符号用空字符取代
15 for ch in songLower:
16     if ch in ".,?":
17         songLower = songLower.replace(ch,'')
18 print("不再有标点符号的歌曲")
19 print(songLower)
20
21 # 将歌曲字符串转成列表
22 songList = songLower.split()
23 print("以下是歌曲列表")
24 print(songList)                      # 打印歌曲列表
25
26 # 将歌曲列表处理成字典
27 for wd in songList:
28     if wd in mydict:                 # 检查此字是否已在字典内
29         mydict[wd] += 1              # 累计出现次数
30     else:
31         mydict[wd] = 1               # 第一次出现的字建立此键与值
32
33 print("以下是最后执行结果")
34 print(mydict)                        # 打印字典
```

执行结果

```
====================== RESTART: D:\Python\ch9\ch9_31.py ======================
原始歌曲
Are you sleeping, are you sleeping, Brother John, Brother John?
Morning bells are ringing, morning bells are ringing.
Ding ding dong, Ding ding dong.
小写歌曲
are you sleeping, are you sleeping, brother john, brother john?
morning bells are ringing, morning bells are ringing.
ding ding dong, ding ding dong.
不再有标点符号的歌曲
are you sleeping are you sleeping brother john brother john
morning bells are ringing morning bells are ringing
ding ding dong ding ding dong
以下是歌曲列表
['are', 'you', 'sleeping', 'are', 'you', 'sleeping', 'brother', 'john', 'brother',
'john', 'morning', 'bells', 'are', 'ringing', 'morning', 'bells', 'are', 'ringing'
, 'ding', 'ding', 'dong', 'ding', 'ding', 'dong']
以下是最后执行结果
{'are': 4, 'you': 2, 'sleeping': 2, 'brother': 2, 'john': 2, 'morning': 2, 'bells'
: 2, 'ringing': 2, 'ding': 4, 'dong': 2}
```

上述程序其实注释非常清楚，整个程序依据下列方式处理。

（1）将歌曲全部改成小写字母同时打印，可参考 10 ～ 12 行。

（2）将歌曲的标点符号“,.?”全部改为空白同时打印，可参考 15 ～ 19 行。

（3）将歌曲字符串转成列表同时打印列表，可参考 22 ～ 24 行。

（4）将歌曲列表处理成字典同时计算每个单字出现次数，可参考 27 ～ 31 行。

（5）最后打印字典。

9-9-2　字典生成式

在 7-2-5 节和 7-2-7 节有介绍列表生成的概念，其实可以将该概念应用在字典生成式，此时语法如下：

新字典 = { 键表达式 : 值表达式 for 表达式 in 可迭代对象 }

程序实例 ch9_32.py：使用字典生成式记录单词 deepstone 中每个字母出现的次数。

```
1  # ch9_32.py
2  word = 'deepstone'
3  alphabetCount = {alphabet:word.count(alphabet) for alphabet in word}
4  print(alphabetCount)
```

执行结果

```
====================== RESTART: D:\Python\ch9\ch9_32.py ======================
{'d': 1, 'e': 3, 'p': 1, 's': 1, 't': 1, 'o': 1, 'n': 1}
```

很不可思议，只需一行程序代码（第 3 行）就将一个单词每个字母的出现次数列出来，这就是 Python 奥妙的地方。上述程序的执行原理是将每个字母出现的次数当作是的值，其实这是真正懂 Python 的程序设计师会使用的方式。当然如果硬要挑上述程序的缺点，就在于对字母 e 而言，在 for 循环中会被执行 3 次，第 10 章会介绍集合（set），将改良这个程序，让读者迈向 Python 高手之路。

当你了解了上述 ch9_32.py 后，若是再看 ch9_31.py 可以发现，第 27 ～ 31 行是将列表改为字典同时计算每个字母的出现次数，该程序花了 5 行处理这个功能，其实可以使用 1 行就取代原先需要 5 行的程序。

程序实例 ch9_33.py：使用列表生成方式重新设计 ch9_31.py，这个程序的重点是第 27 行取代了原先的第 27 ～ 31 行。

```
27  mydict = {wd:songList.count(wd) for wd in songList}
```

另外，可以省略第 5 行设置空字典。

```
5  #mydict = {}                    # 省略,空字典未來存储單字計數結果
```

执行结果 与 ch9_31.py 相同。

9-9-3　设计季节的英汉字典

其实对读者而言这是一个简单的应用，这个程序在执行时会要求输入季节的英文，如果所输入的单词在字典内则输出此单词的中文，如果所输入的单词不在字典则输出查无此单词。

程序实例 ch9_34.py：季节英汉字典的设计。

```
1  # ch9_34.py
2  season = {'Spring':'春季',
3            'Summer':'夏季',
4            'Fall':'秋季',
5            'Winter':'冬季'}
6
7  wd = input("请输入要查询的单词 : ")
8  if wd in season:
9      print(wd, " 中文字义是 : ", season[wd])
10 else:
11     print("查无此单词")
```

执行结果

```
===================== RESTART: D:\Python\ch9\ch9_34.py =====================
请输入要查询的单词 : Spring
Spring  中文字义是 :  春季
>>>
===================== RESTART: D:\Python\ch9\ch9_34.py =====================
请输入要查询的单词 : Table
查无此单词
```

9-9-4　文件加密

延续 6-13-2 节的内容，在 Python 数据结构中，要执行加密可以使用字典的功能，即将原始字符当作键（key），加密结果当作值（value），这样就可以达到加密的目的。若是要让字母往前移 3 个字符，相当于要建立下列字典。

encrypt = {'a':'x', 'b':'y', 'c':'z', 'd':'a', … , 'z':'w'}

程序实例 ch9_35.py：设计一个加密程序，使用“python”做测试。

```
1  # ch9_35.py
2  abc = 'abcdefghijklmnopqrstuvwxyz'
3  encry_dict = {}
4  front3 = abc[:3]
5  end23 = abc[3:]
6  subText = end23 + front3
7  encry_dict = dict(zip(subText, abc))     # 建立字典
8  print("打印编码字典\n", encry_dict)       # 打印字典
9
10 msgTest = 'python'                       # 测试字符串
11 cipher = []
12 for i in msgTest:                        # 执行每个字符加密
13     v = encry_dict[i]                    # 加密
14     cipher.append(v)                     # 加密结果
15 ciphertext = ''.join(cipher)             # 将列表转成字符串
16
17 print("原始字符串 ", msgTest)
18 print("加密字符串 ", ciphertext)
```

执行结果

```
===================== RESTART: D:\Python\ch9\ch9_35.py =====================
打印编码字典
 {'d': 'a', 'e': 'b', 'f': 'c', 'g': 'd', 'h': 'e', 'i': 'f', 'j': 'g', 'k': 'h',
'l': 'i', 'm': 'j', 'n': 'k', 'o': 'l', 'p': 'm', 'q': 'n', 'r': 'o', 's': 'p', 't
': 'q', 'u': 'r', 'v': 's', 'w': 't', 'x': 'u', 'y': 'v', 'z': 'w', 'a': 'x', 'b':
'y', 'c': 'z'}
原始字符串  python
加密字符串  mvqelk
```

在第 12 章会扩充上述程序成可以处理加密整段文件，同时也将讲解文件解密。

习题

1. 请建立星期信息的英汉字典，相当于输入英文的星期信息可以列出星期的中文，如果输入的不是星期的英文则列出输入错误。这个程序的另一个特点是，不论输入大小写均可以处理。（9-1 节）

```
======================= RESTART: D:\Python\ex\ex9_1.py =======================
请输入星期几的英文 : Sunday
星期天
>>>
======================= RESTART: D:\Python\ex\ex9_1.py =======================
请输入星期几的英文 : SUNDAY
星期天
>>>
======================= RESTART: D:\Python\ex\ex9_1.py =======================
请输入星期几的英文 : sunday
星期天
>>>
======================= RESTART: D:\Python\ex\ex9_1.py =======================
请输入星期几的英文 : march
输入错误
```

2. 请建立信息的汉英字典，相当于输入中文的月份（例如：一月）信息可以列出英文的月份，如果输入不是月份的中文则列出输入错误。（9-1 节）

```
======================= RESTART: D:\Python\ex\ex9_2.py =======================
请输入月份(例如:一月) : 六月
June
>>>
======================= RESTART: D:\Python\ex\ex9_2.py =======================
请输入月份(例如:一月) : 月
输入错误
```

3. 有一个 fruits 字典内含 5 种水果的每斤售价，Watermelon:15、Banana:20、Pineapple:25、Orange:12、Apple:18，请先打印此 fruits 字典，再依水果名排序打印。（9-2 节）

```
======================= RESTART: D:\Python\ex\ex9_3.py =======================
{'Watermelon': 15, 'Banana': 20, 'Pineapple': 25, 'Orange': 12, 'Apple': 18}
Apple : 18
Banana : 20
Orange : 12
Pineapple : 25
Watermelon : 15
```

4. 重新设计 ch9_21_1.py，请先打印此 noodles 字典，请设计程序时不使用列表，直接依 noodles 售价排序打印。（9-2 节）

```
======================= RESTART: D:\Python\ex\ex9_4.py =======================
{'牛肉面': 100, '肉丝面': 80, '阳春面': 60, '大卤面': 90, '麻酱面': 70}
阳春面 : 60
麻酱面 : 70
肉丝面 : 80
大卤面 : 90
牛肉面 : 100
```

5. 请使用 max( ) 和 min( ) 方法设计 ch9_21_1.py，打印完 noodles 字典后，直接打印最贵和最便宜的面。（9-2 节）

```
======================= RESTART: D:\Python\ex\ex9_5.py =======================
{'牛肉面': 100, '肉丝面': 80, '阳春面': 60, '大卤面': 90, '麻酱面': 70}
最贵的是   牛肉面 金额是 100
最便宜的是 阳春面 金额是 60
```

6. 重新设计 ch9_24.py，将最后 3 名小兵改成 tag 是 green、score 是 10、speed 是 fast。(9-3 节)

```
====================== RESTART: D:\Python\ex\ex9_6.py ======================
前3名小兵资料
{'tag': 'red', 'score': 3, 'speed': 'slow'}
{'tag': 'red', 'score': 3, 'speed': 'slow'}
{'tag': 'red', 'score': 3, 'speed': 'slow'}
打印编号35到40小兵数据
{'tag': 'red', 'score': 3, 'speed': 'slow'}
{'tag': 'blue', 'score': 5, 'speed': 'medium'}
{'tag': 'blue', 'score': 5, 'speed': 'medium'}
{'tag': 'blue', 'score': 5, 'speed': 'medium'}
{'tag': 'red', 'score': 3, 'speed': 'slow'}
{'tag': 'red', 'score': 3, 'speed': 'slow'}
打印编号47到49小兵数据
{'tag': 'green', 'score': 10, 'speed': 'fast'}
{'tag': 'green', 'score': 10, 'speed': 'fast'}
{'tag': 'green', 'score': 10, 'speed': 'fast'}
```

7. 请参考 ch9_26.py，设计 5 个旅游地点当作键，值则是由字典组成的，内部包含 5 个 " 键 : 值 "，请自行发挥创意，然后打印出来。(9-5 节)

```
====================== RESTART: D:\Python\ex\ex9_7.py ======================
旅游地点 =  张家界
省份     =  湖南省
景点     =  天门山, 大峡谷
旅游地点 =  九寨沟
省份     =  四川省
景点     =  熊猫海, 箭竹海
旅游地点 =  黄山
省份     =  安徽省
景点     =  天都峰, 蓬莱三岛
旅游地点 =  武夷山
省份     =  福建省
景点     =  天游峰, 桃源洞
旅游地点 =  敦煌
省份     =  甘肃省
景点     =  石窟, 月牙泉
```

8. 请扩充设计专题 ch9_32.py，该程序所输出的部分可以不用再输出，本程序会使用所建立的字典，打印出现最多的单词，同时打印出现次数，可能会有多个单词出现一样次数是最多次，必须同时列出来。(9-9 节)

```
====================== RESTART: D:/Python/ex/ex9_8.py ======================
字串 are 出現最多次共出現 4 次
字串 ding 出現最多次共出現 4 次
```

9. 在 Python Shell 环境若是输入 import this，可以看到美国著名软件工程师 Tim Peters 所写的 Python 设计原则 20 则，其实只有 19 则，我们也称之为 Python 之禅（The Zen of Python），如下所示。(9-9 节)

```
>>> import this
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

请设计程序用排序方式列出上述所有单词，以及单词所出现的次数。

```
======================= RESTART: D:/Python/ex/ex9_9.py =======================
a : 2
although : 3
ambiguity : 1
and : 1
are : 1
aren't : 1
at : 1
bad : 1
be : 3
beats : 1
beautiful : 1
better : 8
break : 1
by : 1
cases : 1
```

……

```
special : 2
temptation : 1
than : 8
that : 1
the : 6
there : 1
those : 1
tim : 1
to : 5
ugly : 1
unless : 2
way : 2
you're : 1
zen : 1
```

10. 请扩充 ch9_35.py，处理成可以加密英文大小写，基本思想是让 abc 字符串是 'abcdefghijklmnopqrstuvwxyz ABCDEFGHIJKLMNOPQRSTUVWXYZ'。另外，让 z 和 A 之间空一格，这是让空格也执行加密。这时 a 将加密为 X、b 将加密为 Y、c 将加密为 Z。(9-9 节)

```
====================== RESTART: D:\Python\ex\ex9_10.py ======================
打印编码字典
 {'d': 'a', 'e': 'b', 'f': 'c', 'g': 'd', 'h': 'e', 'i': 'f', 'j': 'g', 'k': 'h'
, 'l': 'i', 'm': 'j', 'n': 'k', 'o': 'l', 'p': 'm', 'q': 'n', 'r': 'o', 's': 'p'
, 't': 'q', 'u': 'r', 'v': 's', 'w': 't', 'x': 'u', 'y': 'v', 'z': 'w', ' ': 'x'
, 'A': 'y', 'B': 'z', 'C': ' ', 'D': 'A', 'E': 'B', 'F': 'C', 'G': 'D', 'H': 'E'
, 'I': 'F', 'J': 'G', 'K': 'H', 'L': 'I', 'M': 'J', 'N': 'K', 'O': 'L', 'P': 'M'
, 'Q': 'N', 'R': 'O', 'S': 'P', 'T': 'Q', 'U': 'R', 'V': 'S', 'W': 'T', 'X': 'U'
, 'Y': 'V', 'Z': 'W', 'a': 'X', 'b': 'Y', 'c': 'Z'}
原始字符串  I like python
加密字符串  Fxifhbxmvqelk
```

10

第 10 章

集合

本章摘要

集合的基本概念是无序且每个元素是**唯一**的，其实也可以将集合看成是字典的键，每个键都是唯一的，集合元素的内容是不可变的（immutable）。常见的元素有**整数**（integer）、**浮点数**（float）、**字符串**（string）、**元组**（tuple）等。至于可变（mutable）内容**列表**（list）、**字典**（dict）、**集合**（set）等不可以是集合元素。但是集合本身是**可变的**（mutable），可以**增加或删除**集合的元素。

10-1　建立集合

Python 可以使用大括号“{ }”或 set( ) 函数建立集合，下面将分别说明。

10-1-1　使用大括号建立集合

Python 允许我们直接使用大括号“{ }”设置集合，例如，如果集合名称是 langs，内容是 'Python' 'C' 'Java'。可以使用下列方式设置集合。

程序实例 ch10_1.py：基本集合的建立。

```
1  # ch10_1.py
2  langs = {'Python', 'C', 'Java'}
3  print("打印集合 = ", langs)
4  print("打印类别 = ", type(langs))
```

执行结果

```
==================== RESTART: D:\Python\ch10\ch10_1.py ====================
打印集合 =  {'Java', 'C', 'Python'}
打印类别 =  <class 'set'>
```

集合的特点是元素是唯一的，所以如果设置集合时有重复元素情形，多的部分将被舍去。

程序实例 ch10_2.py：基本集合的建立，建立时部分元素重复，观察执行结果。

```
1  # ch10_2.py
2  langs = {'Python', 'C', 'Java', 'Python', 'C'}
3  print(langs)
```

执行结果

```
==================== RESTART: D:\Python\ch10\ch10_2.py ====================
{'Python', 'C', 'Java'}
```

上述 'Python' 和 'C' 在设置时都出现两次，但是列出时有重复的元素将只保留一份。集合内容可以是由不同数据类型组成，可参考下列实例。

程序实例 ch10_3.py：使用整数和不同数据类型所建的集合。

```
1  # ch10_3.py
2  # 集合由整数所组成
3  integer_set = {1, 2, 3, 4, 5}
4  print(integer_set)
5  # 集合由不同数据类型所组成
6  mixed_set = {1, 'Python', (2, 5, 10)}
7  print(mixed_set)
8  # 集合的元素是不可变的所以程序第6行所设置的元组元素改成
9  # 第10行列表的写法将会产生错误
10 # mixed_set = { 1, 'Python', [2, 5, 10]}
```

执行结果

```
==================== RESTART: D:\Python\ch10\ch10_3.py ====================
{1, 2, 3, 4, 5}
{1, (2, 5, 10), 'Python'}
```

读者可以将第 10 行的 "#" 删除，可以发现程序会有错误产生，原因是 [2, 5, 10] 是列表，这是可变的元素，所以不可以当作集合元素。

读者可能会思考，**字典**是用大括号定义，**集合**也是用大括号定义，可否直接使用**空**的大括号定义空集合？可参考下列实例。

程序实例 ch10_4.py：建立空集合并观察执行结果，发现错误的实例。

```
1  # ch10_4.py
2  x = {}                        # 这是建立空字典非空集合
3  print("打印     = ", x)
4  print("打印类别 = ", type(x))
```

执行结果

```
==================== RESTART: D:\Python\ch10\ch10_4.py ====================
打印     =  {}
打印类别 =  <class 'dict'>
```

结果发现使用空的大括号 { } 定义，获得的是空字典，10-1-2 节将会讲解定义空集合的方法。

10-1-2　使用 set( ) 函数定义集合

除了用 10-1-1 节的方式建立集合，也可以使用内建的 set( ) 函数建立集合。set( ) 函数参数的内容可以是**字符串**、**列表**、**元组**、**字典**等。这时原先的**字符串**、**列表**、**元组**的元素将被转成集合元素，字典则是键（key）会被转成集合元素。首先回到建立空集合的主题，如果想建立空集合需使用 set( ) 函数。

程序实例 ch10_5.py：重新设计 ch10_4.py，使用 set( ) 函数建立空集合。

```
1  # ch10_5.py
2  empty_dict = {}                    # 这是建立空字典
3  print("打印类别 = ", type(empty_dict))
4  empty_set = set()                  # 这是建立空集合
5  print("打印类别 = ", type(empty_set))
```

执行结果

```
==================== RESTART: D:\Python\ch10\ch10_5.py ====================
打印类别 =  <class 'dict'>
打印类别 =  <class 'set'>
```

程序实例 ch10_6.py：使用字符串（string）建立与打印集合，同时列出集合的数据类型。

```
1  # ch10_6.py
2  x = set('DeepStone mean Deep Learning')
3  print(x)
4  print(type(x))
```

执行结果

```
===================== RESTART: D:\Python\ch10\ch10_6.py =====================
{'m', 'g', 'D', 't', 'r', 'a', 'o', 'p', 'e', ' ', 'i', 'L', 'S', 'n'}
<class 'set'>
```

由于集合元素具有唯一的特性，所以程序第 2 行原先的字符串有许多字母（例如 e）重复，经过 set( ) 处理后，所有英文字母将没有重复。

程序实例 ch10_7.py：使用列表建立与打印集合。

```
1  # ch10_7.py
2  # 表达方式1
3  fruits = ['apple', 'orange', 'apple', 'banana', 'orange']
4  x = set(fruits)
5  print(x)
6  # 表达方式2
7  y = set(['apple', 'orange', 'apple', 'banana', 'orange'])
8  print(y)
```

执行结果

```
===================== RESTART: D:\Python\ch10\ch10_7.py =====================
{'banana', 'apple', 'orange'}
{'banana', 'apple', 'orange'}
```

读者需留意两种不同的 set( ) 函数的使用方式，同时原先列表的内容已经变为集合元素内容了。

程序实例 ch10_8.py：使用元组建立与打印集合。

```
1  # ch10_8.py
2  cities = set(('Beijing', 'Tokyo', 'Beijing', 'Taipei', 'Tokyo'))
3  print(cities)
```

执行结果

```
===================== RESTART: D:\Python\ch10\ch10_8.py =====================
{'Taipei', 'Beijing', 'Tokyo'}
```

程序实例 ch10_8_1.py：使用字典建立集合时，字典的键会被当作集合的元素，这个程序会打印集合。

```
1  # ch10_8_1.py
2  asia = {'China':'Beijing', 'Japan':'Tokyo', 'Thailand':'Bangkok'}
3  asiaSet = set(asia)
4  print(asiaSet)
```

执行结果

```
==================== RESTART: D:/Python/ch10/ch10_8_1.py ====================
{'China', 'Japan', 'Thailand'}
```

10-1-3　大数据与集合的应用

笔者的朋友在某知名企业工作，收集了海量数据使用列表保存，这里面有些数据是重复出现

的，他曾经询问笔者应如何将重复的数据删除，笔者告知如果使用 C 语言可能需花几小时解决，但是如果了解 Python 的集合，只要花约 1 分钟就解决了。其实只要将列表数据使用 set( ) 函数转为集合数据，再使用 list( ) 函数将集合数据转为列表数据就可以了。

程序实例 ch10_9.py：将列表内重复的数据删除。

```
1  # ch10_9.py
2  fruits1 = ['apple', 'orange', 'apple', 'banana', 'orange']
3  x = set(fruits1)                # 将列表转成集合
4  fruits2 = list(x)               # 将集合转成列表
5  print("原先列表数据fruits1 = ", fruits1)
6  print("新的列表数据fruits2 = ", fruits2)
```

执行结果

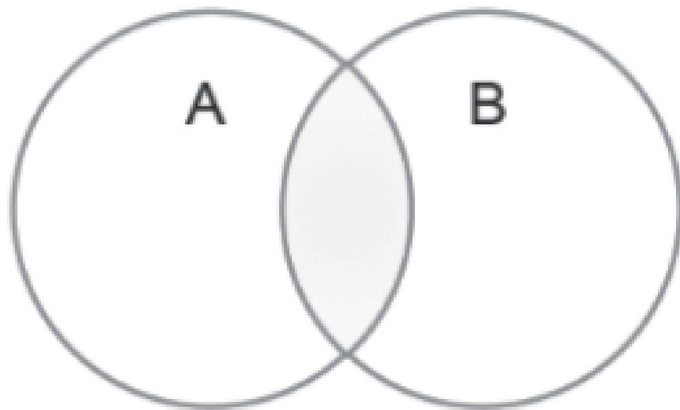
```
==================== RESTART: D:\Python\ch10\ch10_9.py ====================
原先列表数据fruits1 =  ['apple', 'orange', 'apple', 'banana', 'orange']
新的列表数据fruits2 =  ['orange', 'apple', 'banana']
```

10-2 集合的操作

Python 符号	说明
&	交集
|	联集
-	差集
^	对称差集
==	等于
!=	不等于
in	是成员
not in	不是成员

10-2-1 交集

有 A 和 B 两个集合，如果想获得相同的元素，则可以使用**交集**。例如，你举办了数学（**可想成 A 集合**）与物理（**可想成 B 集合**）两个夏令营，如果想统计有哪些人同时参加这两个夏令营，可以使用此功能。



在 Python 语言中的**交集符号**是“&”，另外，也可以使用 intersection( ) 方法完成这个工作。

程序实例 ch10_10.py：有数学与物理两个夏令营，这个程序会列出同时参加这两个夏令营的成员。

```
1  # ch10_10.py
2  math = {'Kevin', 'Peter', 'Eric'}        # 设置参加数学夏令营成员
3  physics = {'Peter', 'Nelson', 'Tom'}     # 设置参加物理夏令营成员
4  both = math & physics
5  print("同时参加数学与物理夏令营的成员 ",both)
```

执行结果

```
==================== RESTART: D:\Python\ch10\ch10_10.py ====================
同时参加数学与物理夏令营的成员  {'Peter'}
```

程序实例 ch10_11.py：使用 intersection( ) 方法完成交集的应用。

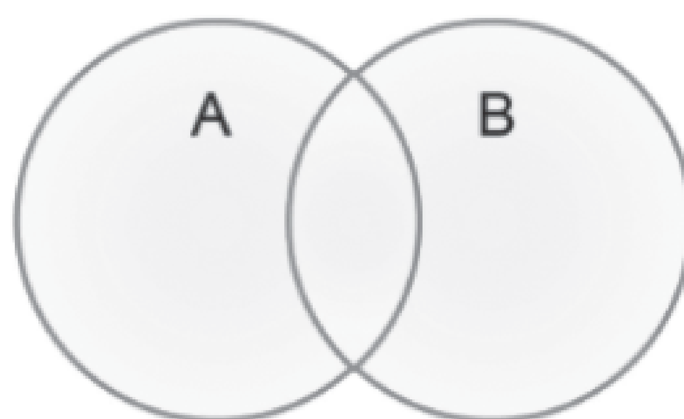
```
1  # ch10_11.py
2  A = {1, 2, 3, 4, 5}           # 定义集合A
3  B = {3, 4, 5, 6, 7}           # 定义集合B
4  # 将intersection()应用在A集合
5  AB = A.intersection(B)        # A和B的交集
6  print("A和B的交集是 ", AB)
7  # 将intersection()应用在B集合
8  BA = B.intersection(A)        # B和A的交集
9  print("B和A的交集是 ", BA)
```

执行结果

```
==================== RESTART: D:\Python\ch10\ch10_11.py ====================
A和B的交集是  {3, 4, 5}
B和A的交集是  {3, 4, 5}
```

10-2-2　联集

有 A 和 B 两个集合，如果想获得所有的元素，则可以使用**联集**。例如，你举办了数学（**可想成 A 集合**）与物理（**可想成 B 集合**）两个夏令营，如果想统计参加这两个夏令营的全部成员，可以使用此功能。



在 Python 语言中的**联集符号**是“|”，另外，也可以使用 union( ) 方法完成这个工作。

程序实例 ch10_12.py：有数学与物理两个夏令营，这个程序会列出参加这两个夏令营的所有成员。

```
1  # ch10_12.py
2  math = {'Kevin', 'Peter', 'Eric'}        # 设置参加数学夏令营成员
3  physics = {'Peter', 'Nelson', 'Tom'}     # 设置参加物理夏令营成员
4  allmember = math | physics
5  print("同时参加数学与物理夏令营的成员 ",allmember)
```

执行结果

```
==================== RESTART: D:\Python\ch10\ch10_12.py ====================
同时参加数学与物理夏令营的成员  {'Nelson', 'Eric', 'Kevin', 'Tom', 'Peter'}
```

程序实例 ch10_13.py：使用 union( ) 方法完成联集的应用。

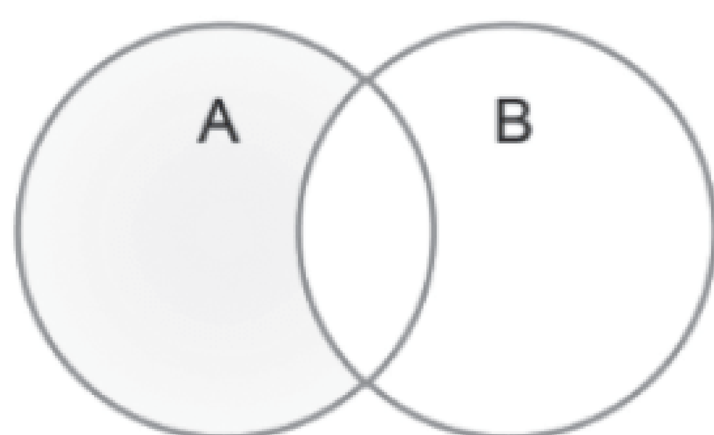
```
1  # ch10_13.py
2  A = {1, 2, 3, 4, 5}              # 定义集合A
3  B = {3, 4, 5, 6, 7}              # 定义集合B
4  # 将union()应用在A集合
5  AorB = A.union(B)                # A和B的联集
6  print("A和B的联集是 ", AorB)
7  # 将union()应用在B集合
8  BorA = B.union(A)                # B和A的联集
9  print("B和A的联集是 ", BorA)
```

执行结果

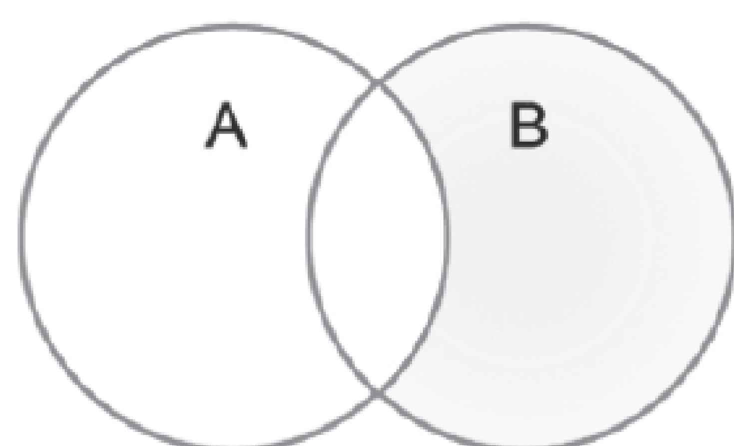
```
==================== RESTART: D:\Python\ch10\ch10_13.py ====================
A和B的联集是  {1, 2, 3, 4, 5, 6, 7}
B和A的联集是  {1, 2, 3, 4, 5, 6, 7}
```

10-2-3　差集

有 A 和 B 两个集合，如果想获得属于 A 集合，同时不属于 B 集合的元素，则可以使用**差集**（A-B）。如果想获得属于 B 集合，同时不属于 A 集合的元素，则可以使用**差集**（B-A）。例如，你举办了数学（**可想成 A 集合**）与物理（**可想成 B 集合**）两个夏令营，如果想了解参加数学夏令营但是没有参加物理夏令营的成员，可以使用此功能。



如果想统计参加物理夏令营但是没有参加数学夏令营的成员，也可以使用此功能。



在 Python 语言中的**差集符号**是 "-"，另外，也可以使用 difference( ) 方法完成这个工作。

程序实例 ch10_14.py：有数学与物理两个夏令营，这个程序会列出参加数学夏令营但是没有参加物理夏令营的所有成员。另外，也会列出参加物理夏令营但是没有参加数学夏令营的所有成员。

```
1  # ch10_14.py
2  math = {'Kevin', 'Peter', 'Eric'}        # 设置参加数学夏令营成员
3  physics = {'Peter', 'Nelson', 'Tom'}     # 设置参加物理夏令营成员
4  math_only = math - physics
5  print("参加数学夏令营同时没有参加物理夏令营的成员 ",math_only)
6  physics_only = physics - math
7  print("参加数学夏令营同时没有参加物理夏令营的成员 ",physics_only)
```

执行结果

```
==================== RESTART: D:\Python\ch10\ch10_14.py ====================
参加数学夏令营同时没有参加物理夏令营的成员  {'Eric', 'Kevin'}
参加数学夏令营同时没有参加物理夏令营的成员  {'Tom', 'Nelson'}
```

程序实例 ch10_15.py：使用 difference( ) 方法完成 A-B 差集与 B-A 差集的应用。

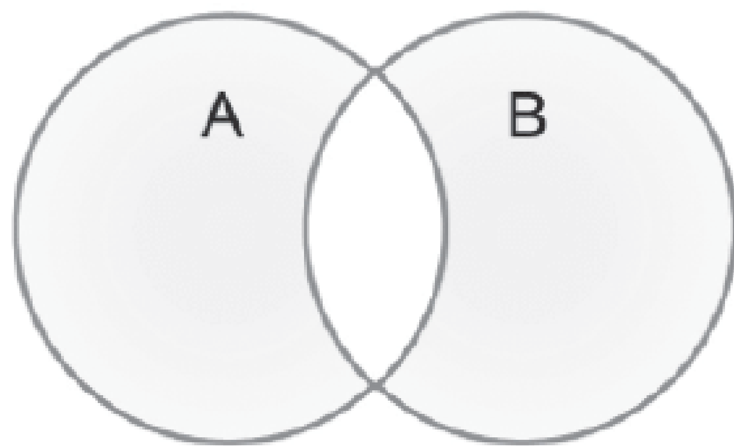
```
1  # ch10_15.py
2  A = {1, 2, 3, 4, 5}                # 定义集合A
3  B = {3, 4, 5, 6, 7}                # 定义集合B
4  # 将difference()应用在A集合
5  A_B = A.difference(B)              # A-B的差集
6  print("A-B的差集是 ", A_B)
7  # 将difference()应用在B集合
8  B_A = B.difference(A)              # B-A的差集
9  print("B-A的差集是 ", B_A)
```

执行结果

```
===================== RESTART: D:\Python\ch10\ch10_15.py =====================
A-B的差集是  {1, 2}
B-A的差集是  {6, 7}
```

10-2-4　对称差集

有 A 和 B 两个集合，如果想获得属于 A 或是 B 集合，但是排除同时属于 A 和 B 的元素，可使用对称差集。例如，你举办了数学（**可想成 A 集合**）与物理（**可想成 B 集合**）两个夏令营，如果想统计参加数学夏令营或是参加物理夏令营的成员，但是排除同时参加这两个夏令营的成员，则可以使用此功能。更简单的解释是只参加一个夏令营的成员。



在 Python 语言中的**对称差集符号是 "^"**，另外，也可以使用 symmetric_difference( ) 方法完成这个工作。

程序实例 ch10_16.py：有数学与物理两个夏令营，这个程序会列出参加数学夏令营或是参加物理夏令营，但是排除同时参加两个夏令营的所有成员。

```
1  # ch10_16.py
2  math = {'Kevin', 'Peter', 'Eric'}        # 设置参加数学夏令营成员
3  physics = {'Peter', 'Nelson', 'Tom'}     # 设置参加物理夏令营成员
4  math_sydi_physics = math ^ physics
5  print("没有同时参加数学和物理夏令营的成员 ",math_sydi_physics)
```

执行结果

```
===================== RESTART: D:\Python\ch10\ch10_16.py =====================
没有同时参加数学和物理夏令营的成员  {'Nelson', 'Kevin', 'Eric', 'Tom'}
```

程序实例 ch10_17.py：使用 symmetric_difference( ) 方法完成 A 和 B 与 B 和 A 对称差集的应用。

```
1  # ch10_17.py
2  A = {1, 2, 3, 4, 5}                       # 定义集合A
3  B = {3, 4, 5, 6, 7}                       # 定义集合B
4  # 将symmetric_difference()应用在A集合
5  A_sydi_B = A.symmetric_difference(B)      # A和B的对称差集
6  print("A和B的对称差集是 ", A_sydi_B)
7  # 将symmetric_difference()应用在B集合
8  B_sydi_A = B.symmetric_difference(A)      # B和A的对称差集
9  print("B和A的对称差集是 ", B_sydi_A)
```

执行结果

```
==================== RESTART: D:\Python\ch10\ch10_17.py ====================
A和B的对称差集是  {1, 2, 6, 7}
B和A的对称差集是  {1, 2, 6, 7}
```

10-2-5　等于

等于的 Python 符号是“==”，可以获得两个集合是否相等，如果相等返回 True，否则返回 False。

程序实例 ch10_18.py：测试两个集合是否相等。

```
1  # ch10_18.py
2  A = {1, 2, 3, 4, 5}                  # 定义集合A
3  B = {3, 4, 5, 6, 7}                  # 定义集合B
4  C = {1, 2, 3, 4, 5}                  # 定义集合C
5  # 列出A与B集合是否相等
6  print("A与B集合相等", A == B)
7  # 列出A与C集合是否相等
8  print("A与C集合相等", A == C)
```

执行结果

```
==================== RESTART: D:\Python\ch10\ch10_18.py ====================
A与B集合相等 False
A与C集合相等 True
```

10-2-6　不等于

不等于的 Python 符号是“!=”，可以获得两个集合是否不相等，如果不相等返回 True，否则返回 False。

程序实例 ch10_19.py：测试两个集合是否不相等。

```
1  # ch10_19.py
2  A = {1, 2, 3, 4, 5}                  # 定义集合A
3  B = {3, 4, 5, 6, 7}                  # 定义集合B
4  C = {1, 2, 3, 4, 5}                  # 定义集合C
5  # 列出A与B集合是否相等
6  print("A与B集合不相等", A != B)
7  # 列出A与C集合是否不相等
8  print("A与C集合不相等", A != C)
```

执行结果

```
==================== RESTART: D:\Python\ch10\ch10_19.py ====================
A与B集合不相等 True
A与C集合不相等 False
```

10-2-7　是成员 in

Python 的关键词 in 可以测试元素是否是集合的元素成员。

程序实例 ch10_20.py：关键词 in 的应用。

```
1  # ch10_20.py
2  # 方法1
3  fruits = set("orange")
4  print("字符a是属于fruits集合?", 'a' in fruits)
5  print("字符d是属于fruits集合?", 'd' in fruits)
6  # 方法2
7  cars = {"Nissan", "Toyota", "Ford"}
8  boolean = "Ford" in cars
9  print("Ford in cars", boolean)
10 boolean = "Audi" in cars
11 print("Audi in cars", boolean)
```

执行结果

```
===================== RESTART: D:\Python\ch10\ch10_20.py =====================
字符a是属于fruits集合? True
字符d是属于fruits集合? False
Ford in cars True
Audi in cars False
```

程序实例 ch10_21.py：使用循环列出所有参加数学夏令营的学生。

```
1  # ch10_21.py
2  math = {'Kevin', 'Peter', 'Eric'}        # 设置参加数学夏令营成员
3  print("打印参加数学夏令营的成员")
4  for name in math:
5      print(name)
```

执行结果

```
===================== RESTART: D:\Python\ch10\ch10_21.py =====================
打印参加数学夏令营的成员
Eric
Peter
Kevin
```

10-2-8　不是成员 not in

Python 的关键词 not in 可以测试元素是否不是集合的元素成员。

程序实例 ch10_22.py：关键词 not in 的应用。

```
1  # ch10_22.py
2  # 方法1
3  fruits = set("orange")
4  print("字符a是不属于fruits集合?", 'a' not in fruits)
5  print("字符d是不属于fruits集合?", 'd' not in fruits)
6  # 方法2
7  cars = {"Nissan", "Toyota", "Ford"}
8  boolean = "Ford" not in cars
9  print("Ford not in cars", boolean)
10 boolean = "Audi" not in cars
11 print("Audi not in cars", boolean)
```

执行结果

```
===================== RESTART: D:\Python\ch10\ch10_22.py =====================
字符a是不属于fruits集合? False
字符d是不属于fruits集合? True
Ford not in cars False
Audi not in cars True
```

10-3 适用集合的方法

方法	说明
add( )	加一个元素到集合
clear( )	删除集合所有元素
copy( )	复制集合
difference_update( )	删除集合内与另一集合重复的元素
discard( )	如果是集合成员则删除
intersection_update( )	可以使用交集更新集合内容
isdisjoint( )	如果两个集合没有交集返回 True
issubset( )	如果另一个集合包含这个集合返回 True
isupperset( )	如果这个集合包含另一个集合返回 True
pop( )	返回所删除的元素，如果是空集合返回 False
remove( )	删除指定元素，如果此元素不存在，程序将返回 KeyError
symmetric_difference_update( )	使用对称差集更新集合内容
update( )	使用联集更新集合内容

10-3-1 add( )

add( ) 可以增加一个元素，它的语法格式如下：

集合 A.add(新增元素)

上述语句会将 add( ) 参数的新增元素加到调用此方法的集合 A 内。

程序实例 ch10_22_1.py：在集合内新增元素的应用。

```
1  # ch10_22_1.py
2  cities = { 'Taipei', 'Beijing', 'Tokyo'}
3  # 增加一般元素
4  cities.add('Chicago')
5  print('cities集合内容 ', cities)
6  # 增加已有元素并观察执行结果
7  cities.add('Beijing')
8  print('cities集合内容 ', cities)
9  # 增加元组元素并观察执行结果
10 tup = (1, 2, 3)
11 cities.add(tup)
12 print('cities集合内容 ', cities)
```

执行结果

```
==================== RESTART: D:\Python\ch10\ch10_22_1.py ====================
cities集合内容  {'Beijing', 'Chicago', 'Tokyo', 'Taipei'}
cities集合内容  {'Beijing', 'Chicago', 'Tokyo', 'Taipei'}
cities集合内容  {(1, 2, 3), 'Chicago', 'Beijing', 'Tokyo', 'Taipei'}
```

上述第 7 行，由于集合中已经有 'Beijing' 字符串，将不改变集合 cities 的内容。另外，集合是无序的，可能获得不同的排列结果。

10-3-2　copy( )

集合复制 copy( ) 这个方法不需要参数，相同的概念可以参考 6-8 节，语法格式如下：

新集合名称 = 旧集合名称 .copy( )

程序实例 ch10_23.py：赋值与浅拷贝的比较。

```
1  # ch10_23.py
2  # 赋值
3  numset = {1, 2, 3}
4  deep_numset = numset
5  deep_numset.add(10)
6  print("赋值   - 观察numset        ", numset)
7  print("赋值   - 观察deep_numset   ", deep_numset)
8
9  # 浅拷贝shallow copy
10 shallow_numset = numset.copy()
11 shallow_numset.add(100)
12 print("浅拷贝 - 观察numset        ", numset)
13 print("浅拷贝 - 观察shallow_numset", shallow_numset)
```

执行结果

```
==================== RESTART: D:\Python\ch10\ch10_23.py ====================
赋值   - 观察numset         {10, 1, 2, 3}
赋值   - 观察deep_numset    {10, 1, 2, 3}
浅拷贝 - 观察numset         {10, 1, 2, 3}
浅拷贝 - 观察shallow_numset {1, 2, 3, 100, 10}
```

10-3-3　remove( )

remove( ) 可以删除集合中的指定元素，如果指定删除的元素不存在，将有 KeyError 产生。它的语法格式如下：

集合 A.remove(要删除的元素)

上述语句会将集合 A 内 remove( ) 参数指定的元素删除。

程序实例 ch10_24.py：使用 remove( ) 删除集合元素成功的应用。

```
1  # ch10_24.py
2  countries = {'Japan', 'China', 'France'}
3  print("删除前的countries集合 ", countries)
4  countries.remove('Japan')
5  print("删除后的countries集合 ", countries)
```

执行结果

```
==================== RESTART: D:\Python\ch10\ch10_24.py ====================
删除前的countries集合  {'China', 'Japan', 'France'}
删除后的countries集合  {'China', 'France'}
```

程序实例 ch10_25.py：使用 remove( ) 删除集合元素失败的应用。

```
1  # ch10_25.py
2  animals = {'dog', 'cat', 'bird'}
3  print("删除前的animals集合 ", animals)
4  animals.remove('fish')          # 删除不存在的元素产生错误
5  print("删除后的animals集合 ", animals)
```

执行结果

```
===================== RESTART: D:\Python\ch10\ch10_25.py =====================
删除前的animals集合  {'bird', 'dog', 'cat'}
Traceback (most recent call last):
  File "D:\Python\ch10\ch10_25.py", line 4, in <module>
    animals.remove('fish')          # 删除不存在的元素产生错误
KeyError: 'fish'
```

上述程序中由于 fish 不存在于 animals 集合中，所以会产生错误。如果要避免这类错误，可以使用 discard( ) 方法。

10-3-4 discard( )

discard( ) 可以删除集合中的元素，如果元素不存在也不会有错误产生。

ret_value = 集合 A.discard(要删除的元素)

上述语句会将集合 A 内 discard( ) 参数指定的元素删除。不论删除结果为何，这个方法会返回 None，这个 None 在一些程序语言中其实称为 NULL，11-3 节会介绍更多函数返回值与返回 None 的知识。

程序实例 ch10_26.py：使用 discard( ) 删除集合元素的应用。

```
1  # ch10_26.py
2  animals = {'dog', 'cat', 'bird'}
3  print("删除前的animals集合    ", animals)
4  # 要删除元素有在集合内
5  animals.discard('cat')
6  print("删除后的animals集合    ", animals)
7  # 要删除元素没有在集合内
8  animals.discard('pig')
9  print("删除后的animals集合    ", animals)
10 # 打印返回值
11 print("删除数据存在的返回值   ", animals.discard('dog'))
12 print("删除数据不存在的返回值 ", animals.discard('pig'))
```

执行结果

```
===================== RESTART: D:\Python\ch10\ch10_26.py =====================
删除前的animals集合     {'cat', 'dog', 'bird'}
删除后的animals集合     {'dog', 'bird'}
删除后的animals集合     {'dog', 'bird'}
删除数据存在的返回值    None
删除数据不存在的返回值  None
```

10-3-5 pop( )

pop( ) 是用随机方式删除集合元素，所删除的元素将被返回，如果集合是空集合则程序会产生 TypeError 错误。

ret_element = 集合 A.pop( )

上述语句会随机删除集合 A 内的元素，所删除的元素将被返回 ret_element。

程序实例 ch10_27.py：使用 pop( ) 删除集合元素的应用。

```
1  # ch10_27.py
2  animals = {'dog', 'cat', 'bird'}
3  print("删除前的animals集合 ", animals)
4  ret_element = animals.pop()
5  print("删除后的animals集合 ", animals)
6  print("所删除的元素是       ", ret_element)
```

执行结果

```
==================== RESTART: D:\Python\ch10\ch10_27.py ====================
删除前的animals集合  {'cat', 'dog', 'bird'}
删除后的animals集合  {'dog', 'bird'}
所删除的元素是       cat
```

10-3-6　clear( )

clear( ) 可以删除集合内的所有元素，返回值是 None。

程序实例 ch10_28.py：使用 clear( ) 删除集合所有元素的应用，这个程序会列出删除所有集合元素前后的集合内容，同时也列出删除空集合的结果。

```
1  # ch10_28.py
2  states = {'Mississippi', 'Idoho', 'Florida'}
3  print("删除前的states集合     ", states)
4  states.clear()
5  print("删除前的states集合     ", states)
6
7  # 测试删除空集合
8  empty_set = set()
9  print("删除前的empty_set集合 ", empty_set)
10 states.clear()
11 print("删除前的empty_set集合 ", empty_set)
```

执行结果

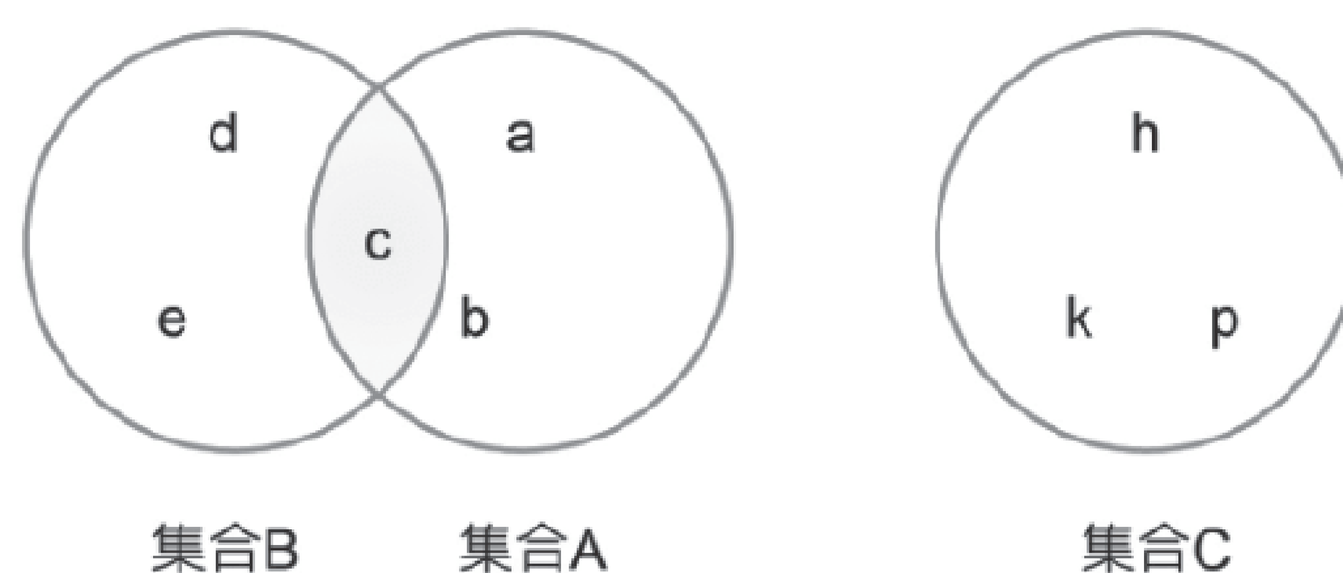
```
==================== RESTART: D:\Python\ch10\ch10_28.py ====================
删除前的states集合     {'Mississippi', 'Idoho', 'Florida'}
删除前的states集合     set()
删除前的empty_set集合  set()
删除前的empty_set集合  set()
```

10-3-7　isdisjoint( )

如果两个集合没有共同的元素会返回 True，否则返回 False。

ret_boolean = 集合 A.isdisjoint(集合 B)

程序实例 ch10_29.py：测试 isdisjoint( )，下列是集合 A，B 和 C 的集合示意图。




```
1  # ch10_29.py
2  A = {'a', 'b', 'c'}
3  B = {'c', 'd', 'e'}
4  C = {'h', 'k', 'p'}
5  # 测试A和B集合
6  boolean = A.isdisjoint(B)       # 有共同的元素'c'
7  print("有共同的元素返回值是    ", boolean)
8
9  # 测试A和C集合
10 boolean = A.isdisjoint(C)       # 没有共同的元素
11 print("没有共同的元素返回值是 ", boolean)
```

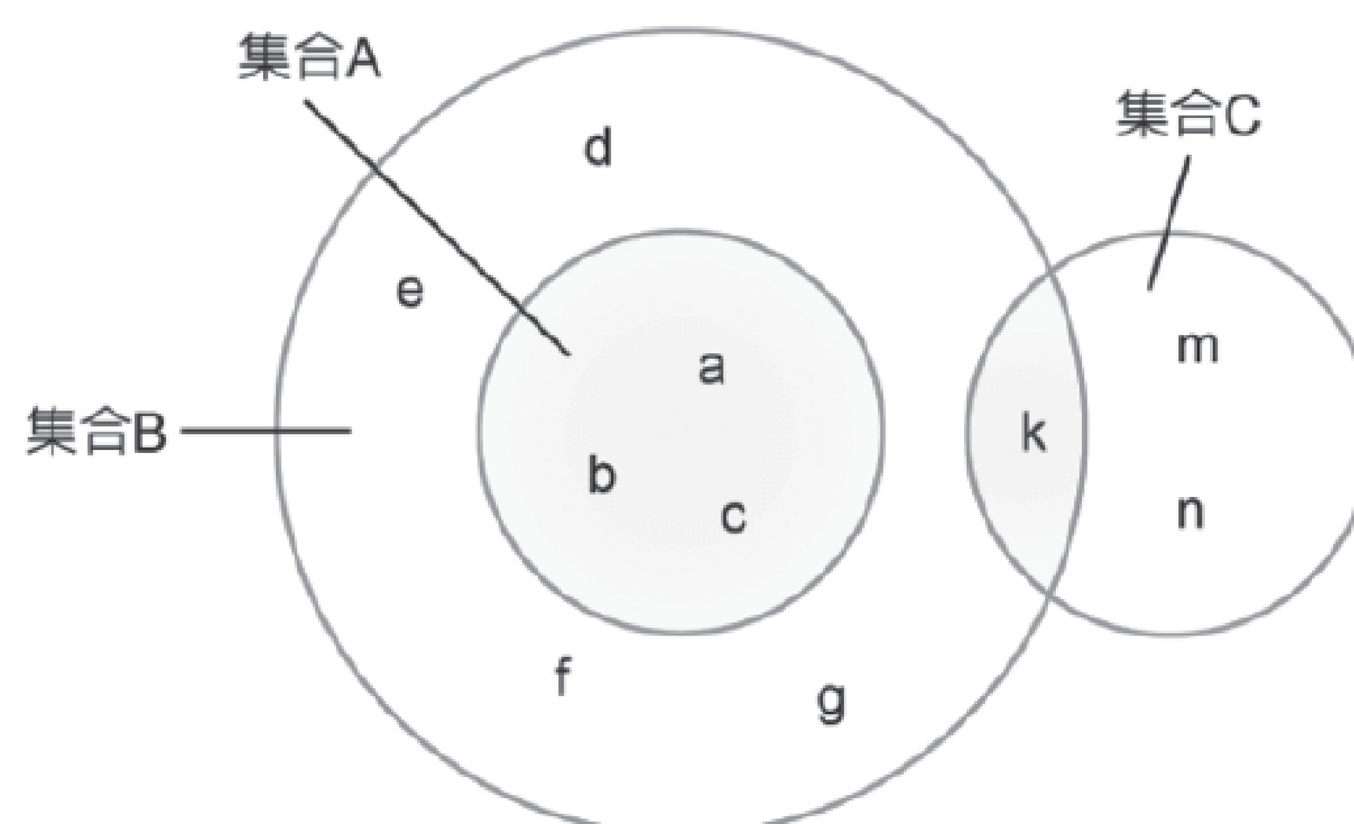
执行结果

```
==================== RESTART: D:\Python\ch10\ch10_29.py ====================
有共同的元素返回值是    False
没有共同的元素返回值是  True
```

10-3-8 issubset( )

这个方法可以测试一个函数是否是另一个函数的子集合，例如，A 集合所有元素均可在 B 集合内发现，则 A 集合是 B 集合的子集合。如果是则返回 True，否则返回 False。

程序实例 ch10_30.py：测试 issubset( )，下列是 A，B 和 C 的集合示意图。




```
1  # ch10_30.py
2  A = {'a', 'b', 'c'}
3  B = {'a', 'b', 'c', 'd', 'e', 'f', 'g', 'k'}
4  C = {'k', 'm', 'n'}
5  # 测试A和B集合
6  boolean = A.issubset(B)          # 所有A的元素都是B的元素
7  print("A集合是B集合的子集合返回值是 ", boolean)
8
9  # 测试C和B集合
10 boolean = C.issubset(B)          # 有共同的元素k
11 print("C集合是B集合的子集合返回值是 ", boolean)
```

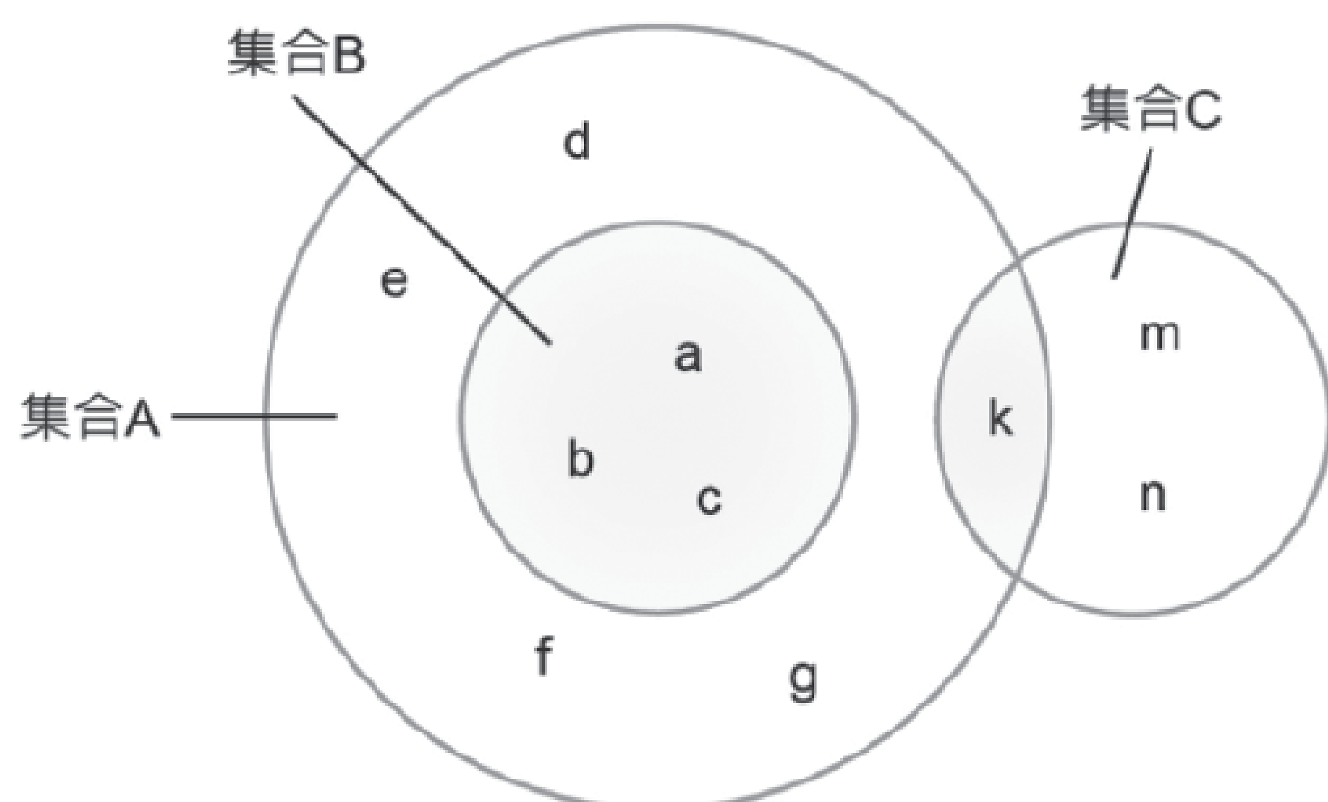
执行结果

```
==================== RESTART: D:\Python\ch10\ch10_30.py ====================
A集合是B集合的子集合返回值是  True
C集合是B集合的子集合返回值是  False
```

10-3-9　issuperset( )

这个方法可以测试一个集合是否是另一个集合的父集合，例如，B 集合所有元素均可在 A 集合内发现，则 A 集合是 B 集合的父集合。如果是则返回 True，否则返回 False。

程序实例 ch10_31.py：测试 issuperset( )，下列是 A，B 和 C 的集合示意图。




```
1  # ch10_31.py
2  A = {'a', 'b', 'c', 'd', 'e', 'f', 'g', 'k'}
3  B = {'a', 'b', 'c'}
4  C = {'k', 'm', 'n'}
5  # 测试A和B集合
6  boolean = A.issuperset(B)           # 测试
7  print("A集合是B集合的父集合返回值是 ", boolean)
8
9  # 测试A和C集合
10 boolean = A.issuperset(C)           # 测试
11 print("A集合是C集合的父集合返回值是 ", boolean)
```

执行结果

```
==================== RESTART: D:\Python\ch10\ch10_31.py ====================
A集合是B集合的父集合返回值是  True
A集合是C集合的父集合返回值是  False
```

10-3-10　intersection_update( )

这个方法将返回集合的交集，它的语法格式如下：

```
ret_value = A.intersection_update(*B)
```

上述语句中 *B 代表可以有一到多个集合，如果只有一个集合，例如是 B，则执行后 A 将是 A 与 B 的交集。如果 *B 代表（B, C），则执行后 A 将是 A、B 与 C 的交集。

上述语句返回值是 None，此值将设置给 ret_value，接下来几节的方法都会返回 None，将不再叙述。

程序实例 ch10_32.py：intersection_update( ) 的应用。

```
1  # ch10_32.py
2  A = {'a', 'b', 'c', 'd'}
3  B = {'a', 'k', 'c'}
4  C = {'c', 'f', 'w'}
5  # A将是A和B的交集
6  ret_value = A.intersection_update(B)
7  print(ret_value)
8  print("A集合 = ", A)
9  print("B集合 = ", B)
10
11 # A将是A, B和C的交集
12 ret_value = A.intersection_update(B, C)
13 print(ret_value)
14 print("A集合 = ", A)
15 print("B集合 = ", B)
16 print("C集合 = ", C)
```

执行结果

```
===================== RESTART: D:\Python\ch10\ch10_32.py =====================
None
A集合 =  {'c', 'a'}
B集合 =  {'k', 'c', 'a'}
None
A集合 =  {'c'}
B集合 =  {'k', 'c', 'a'}
C集合 =  {'w', 'f', 'c'}
```

10-3-11 update( )

可以将一个集合的元素加到调用此方法的集合内，它的语法格式如下：

集合 A.update (集合 B)

上述语句是将集合 B 的元素加到集合 A 内。

程序实例 ch10_33.py：update( ) 的应用。

```
1  # ch10_33.py
2  cars1 = {'Audi', 'Ford', 'Toyota'}
3  cars2 = {'Nissan', 'Toyota'}
4  print("执行update()前列出cars1和cars2内容")
5  print("cars1 = ", cars1)
6  print("cars2 = ", cars2)
7  cars1.update(cars2)
8  print("执行update()后列出cars1和cars2内容")
9  print("cars1 = ", cars1)
10 print("cars2 = ", cars2)
```

执行结果

```
===================== RESTART: D:\Python\ch10\ch10_33.py =====================
执行update()前列出cars1和cars2内容
cars1 =  {'Toyota', 'Audi', 'Ford'}
cars2 =  {'Toyota', 'Nissan'}
执行update()后列出cars1和cars2内容
cars1 =  {'Nissan', 'Toyota', 'Audi', 'Ford'}
cars2 =  {'Toyota', 'Nissan'}
```

10-3-12　difference_update( )

可以删除集合内与另一集合重复的元素，它的语法格式如下：

集合 A.difference_update(集合 B)

上述语句是将集合 A 内与集合 B 重复的元素删除，结果存在 A 集合中。

程序实例 ch10_34.py：difference_update( ) 的应用。执行这个程序后，在集合 A 内与集合 B 重复的元素 Toyota 将被删除。

```
 1  # ch10_34.py
 2  cars1 = {'Audi', 'Ford', 'Toyota'}
 3  cars2 = {'Nissan', 'Toyota'}
 4  print("执行difference_update()前列出cars1和cars2内容")
 5  print("cars1 = ", cars1)
 6  print("cars2 = ", cars2)
 7  cars1.difference_update(cars2)
 8  print("执行difference_update()后列出cars1和cars2内容")
 9  print("cars1 = ", cars1)
10  print("cars2 = ", cars2)
```

执行结果

```
===================== RESTART: D:\Python\ch10\ch10_34.py =====================
执行difference_update()前列出cars1和cars2内容
cars1 =  {'Toyota', 'Audi', 'Ford'}
cars2 =  {'Toyota', 'Nissan'}
执行difference_update()后列出cars1和cars2内容
cars1 =  {'Audi', 'Ford'}
cars2 =  {'Toyota', 'Nissan'}
```

10-3-13　symmetric_difference_update( )

与 10-2-4 节的对称差集一样，只更改了调用此方法的集合。

集合 A.symmetric_difference_update(集合 B)

程序实例 ch10_35.py：symmetric_difference_update( ) 的基本应用。

```
 1  # ch10_35.py
 2  cars1 = {'Audi', 'Ford', 'Toyota'}
 3  cars2 = {'Nissan', 'Toyota'}
 4  print("执行symmetric_difference_update( )前列出cars1和cars2内容")
 5  print("cars1 = ", cars1)
 6  print("cars2 = ", cars2)
 7  cars1.symmetric_difference_update(cars2)
 8  print("执行symmetric_difference_update( )后列出cars1和cars2内容")
 9  print("cars1 = ", cars1)
10  print("cars2 = ", cars2)
```

执行结果

```
===================== RESTART: D:\Python\ch10\ch10_35.py =====================
执行symmetric_difference_update()前列出cars1和cars2内容
cars1 =  {'Ford', 'Audi', 'Toyota'}
cars2 =  {'Nissan', 'Toyota'}
执行symmetric_difference_update()后列出cars1和cars2内容
cars1 =  {'Ford', 'Nissan', 'Audi'}
cars2 =  {'Nissan', 'Toyota'}
```

10-4 适用于集合的基本函数操作

函数名称	说明
enumerate( )	返回连续整数配对的 enumerate 对象
len( )	元素数量
max( )	最大值
min( )	最小值
sorted( )	返回已经排序的列表，集合本身则不改变
sum( )	总和

上述概念与列表或元组相同，本节将不再用实例讲解。

10-5 冻结集合 frozenset

set 是可变集合，frozenset 是不可变集合，也可直译为**冻结集合**，这是一个新的类（class），只要设置元素后，这个冻结集合就不能再更改了。如果将元组想成**不可变列表**（immutable list），冻结集合就是**不可变集合**（immutable set）。

冻结集合的不可变特性的优点是可以用它作为字典的键（key），也可以作为其他集合的元素。冻结集合的建立方式是使用 frozenset( ) 函数。冻结集合建立完成后，不可使用 add( ) 或 remove( ) 更改冻结集合的内容。但是可以执行 intersection( )、union( )、difference( )、symmetric_difference( )、copy( )、issubset( )、issuperset( )、isdisjoint( ) 等方法。

程序实例 ch10_36.py：建立与操作冻结集合。

```
1   # ch10_36.py
2   X = frozenset([1, 3, 5])
3   Y = frozenset([5, 7, 9])
4   print(X)
5   print(Y)
6   print("交集  = ", X & Y)
7   print("联集  = ", X | Y)
8   A = X & Y
9   print("交集A = ", A)
10  A = X.intersection(Y)
11  print("交集A = ", A)
```

执行结果

```
==================== RESTART: D:\Python\ch10\ch10_36.py ====================
frozenset({1, 3, 5})
frozenset({9, 5, 7})
交集  =  frozenset({5})
联集  =  frozenset({1, 3, 5, 7, 9})
交集A =  frozenset({5})
交集A =  frozenset({5})
```

10-6 专题——夏令营程序 / 程序效率 / 集合生成式 / 鸡尾酒实例

10-6-1　夏令营程序设计

程序实例 ch10_37.py：有一个班级有 10 个人，其中有 3 个人参加了数学夏令营，另外有 3 个人参加了物理夏令营，这个程序会列出同时参加数学和物理夏令营的人，同时也会列出有哪些人没有参加暑期夏令营。

```
1  # ch10_37.py
2  # students是学生名单集合
3  students = {'Peter', 'Norton', 'Kevin', 'Mary', 'John',
4              'Ford', 'Nelson', 'Damon', 'Ivan', 'Tom'
5             }
6
7  Math = {'Peter', 'Kevin', 'Damon'}           # 数学夏令营参加人员
8  Physics = {'Nelson', 'Damon', 'Tom' }        # 物理夏令营参加人员
9
10 MandP = Math | Physics
11 print("有 %d 人参加数学和物理夏令营名单  : " % len(MandP), MandP )
12 unAttend = students - MandP
13 print("没有参加任何夏令营有 %d 人名单是 : " % len(unAttend), unAttend)
```

执行结果

```
===================== RESTART: D:\Python\ch10\ch10_37.py =====================
有 5 人参加数学和物理夏令营名单  :  {'Damon', 'Tom', 'Kevin', 'Nelson', 'Peter'}
没有参加任何夏令营有 5 人名单是 :  {'Ivan', 'John', 'Ford', 'Mary', 'Norton'}
```

10-6-2　集合生成式

在先前的章节中已经看过列表和字典的生成式了，其实集合也有生成式，语法如下：

新集合 = { 表达式　for　表达式　in　可迭代项目 }

程序实例 ch10_38.py：产生 1,3, …, 99 的集合。

```
1  # ch10_38.py
2  A = {n for n in range(1,100,2)}
3  print(type(A))
4  print(A)
```

执行结果

```
===================== RESTART: D:/Python/ch10/ch10_38.py =====================
<class 'set'>
{1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31, 33, 35, 37, 39, 41,
43, 45, 47, 49, 51, 53, 55, 57, 59, 61, 63, 65, 67, 69, 71, 73, 75, 77, 79, 81,
83, 85, 87, 89, 91, 93, 95, 97, 99}
```

在集合的生成式中，也可以增加 if 测试句（可以有多个）。

程序实例 ch10_39.py：产生 11,33, …, 99 的集合。

```
1  # ch10_39.py
2  A = {n for n in range(1,100,2) if n % 11 == 0}
3  print(type(A))
4  print(A)
```

执行结果

```
==================== RESTART: D:/Python/ch10/ch10_39.py ====================
<class 'set'>
{33, 99, 11, 77, 55}
```

集合生成式可以让程序设计变得很简洁，例如，过去要建立一系列有规则的序列，先要使用列表生成式，然后将列表改为集合，现在可以直接用集合生成式完成此工作。

10-6-3　提高程序效率

在 ch9_32.py 程序第 3 行的 for 循环如下：

```
for alphabet in word
```

word 的内容是‘deepstone’，在上述循环中将造成字母 e 处理 3 次，其实只要将集合应用于 word，由于集合不会有重复的元素，所以只要处理一次即可，此时可以将上述循环改为：

```
for alphabet in set(word)
```

经上述处理后字母 e 将只执行一次，所以可以提高程序效率。

程序实例 ch10_40.py：使用集合重新设计 ch9_32.py。

```
1  # ch10_40.py
2  word = 'deepstone'
3  alphabetCount = {alphabet:word.count(alphabet) for alphabet in set(word)}
4  print(alphabetCount)
```

执行结果

```
==================== RESTART: D:/Python/ch10/ch10_40.py ====================
{'d': 1, 'o': 1, 'e': 3, 't': 1, 's': 1, 'n': 1, 'p': 1}
```

10-6-4　鸡尾酒的实例

鸡尾酒是酒精饮料，由基酒和一些饮料调制而成，下列是一些常见的鸡尾酒饮料以及它们的配方。

（1）蓝色夏威夷佬（Blue Hawaiian）：兰姆酒（Rum）、甜酒（Sweet Wine）、椰奶（Coconut Cream）、菠萝汁（Pineapple Juice）、柠檬汁（Lemon Juice）。

（2）姜味莫西多（Ginger Mojito）：兰姆酒（Rum）、姜（Ginger）、薄荷叶（Mint Leaves）、莱姆汁（Lime Juice）、姜汁汽水（Ginger Soda）。

（3）纽约客（New Yorker）：威士忌（Whiskey）、红酒（Red Wine）、柠檬汁（Lemon Juice）、糖水（Sugar Syrup）。

（4）血腥玛莉（Bloody Mary）：伏特加（Vodka）、柠檬汁（Lemon Juice）、西红柿汁（Tomato Juice）、酸辣酱（Tabasco）、少量盐（Little Salt）。

程序实例 ch10_41.py：为上述鸡尾酒建立一个字典，字典的键（key）是字符串，也就是鸡尾酒的名称，字典的值是集合，内容是各种鸡尾酒的材料配方。这个程序会列出含有伏特加的酒，含有柠檬汁的酒，含有兰姆酒但没有姜的酒。

```
1  # ch10_41.py
2  cocktail = {
3      'Blue Hawaiian':{'Rum','Sweet Wine','Cream','Pineapple Juice','Lemon Juice'},
4      'Ginger Mojito':{'Rum','Ginger','Mint Leaves','Lime Juice','Ginger Soda'},
5      'New Yorker':{'Whiskey','Red Wine','Lemon Juice','Sugar Syrup'},
6      'Bloody Mary':{'Vodka','Lemon Juice','Tomato Juice','Tabasco','little Sale'}
7      }
8  # 列出含有Vodka的酒
9  print("含有Vodka的酒 : ")
10 for name, formulas in cocktail.items():
11     if 'Vodka' in formulas:
12         print(name)
13 # 列出含有Lemon Juice的酒
14 print("含有Lemon Juice的酒 : ")
15 for name, formulas in cocktail.items():
16     if 'Lemon Juice' in formulas:
17         print(name)
18 # 列出含有Rum但是没有姜的酒
19 print("含有Rum但是没有姜的酒 : ")
20 for name, formulas in cocktail.items():
21     if 'Rum' in formulas and not ('Ginger' in formulas):
22         print(name)
23 # 列出含有Lemon Juice但是没有Cream或是Tabasco的酒
24 print("含有Lemon Juice但是没有Cream或是Tabasco的酒 : ")
25 for name, formulas in cocktail.items():
26     if 'Lemon Juice' in formulas and not formulas & {'Cream', 'Tabasco'}:
27         print(name)
```

执行结果

```
==================== RESTART: D:\Python\ch10\ch10_41.py ====================
含有Vodka的酒 :
Bloody Mary
含有Lemon Juice的酒 :
Blue Hawaiian
New Yorker
Bloody Mary
含有Rum但是没有姜的酒 :
Blue Hawaiian
含有Lemon Juice但是没有Cream或是Tabasco的酒 :
New Yorker
```

上述程序用 in 测试指定的鸡尾酒材料配方是否在所返回字典值（value）的 formulas 集合内，另外，程序第 26 行则是将 formulas 与集合元素 'Cream' 和 'Tabasco' 做交集（&），如果 formulas 内没有这些配方结果会是 False，经过 not 就会是 True，则可以打印 name。

习题

1. 有一段英文如下：(10-1 节)

Silicon Stone Education is an unbiased organization, concentrated on bridging the gap between academic and the working world in order to benefit society as a whole. We have carefully crafted our online certification system and test content databases. The content for each topic is created by experts and is all carefully designed with a comprehensive knowledge to greatly benefit all candidates who participate.

请将上述文章处理成没有标点符号和没有重复字符串的字符串列表。

```
====================== RESTART: D:\Python\ex\ex10_1.py ======================
最后列表 =  ['a', 'academic', 'all', 'an', 'and', 'as', 'benefit', 'between', 'b
ridging', 'by', 'candidates', 'carefully', 'certification', 'comprehensive', 'co
ncentrated', 'content', 'crafted', 'created', 'databases', 'designed', 'each', '
education', 'experts', 'for', 'gap', 'greatly', 'have', 'in', 'is', 'knowledge',
 'on', 'online', 'order', 'organization', 'our', 'participate', 'silicon', 'soci
ety', 'stone', 'system', 'test', 'the', 'to', 'topic', 'unbiased', 'we', 'who',
'whole', 'with', 'working', 'world']
```

2. 请建立两个列表：(10-2 节)

A：1, 3, 5, … , 99

B：0, 5, 10, … , 100

将上述列表转成集合，然后求交集，联集，A-B 差集和 B-A 差集。

```
====================== RESTART: D:\Python\ex\ex10_2.py ======================
联集 :  {0, 1, 3, 5, 7, 9, 10, 11, 13, 15, 17, 19, 20, 21, 23, 25, 27, 29, 30, 3
1, 33, 35, 37, 39, 40, 41, 43, 45, 47, 49, 50, 51, 53, 55, 57, 59, 60, 61, 63, 6
5, 67, 69, 70, 71, 73, 75, 77, 79, 80, 81, 83, 85, 87, 89, 90, 91, 93, 95, 97, 9
9, 100}
交集 :  {65, 35, 5, 75, 45, 15, 85, 55, 25, 95}
A-B差集 :  {1, 3, 7, 9, 11, 13, 17, 19, 21, 23, 27, 29, 31, 33, 37, 39, 41, 43,
47, 49, 51, 53, 57, 59, 61, 63, 67, 69, 71, 73, 77, 79, 81, 83, 87, 89, 91, 93,
97, 99}
B-A差集 :  {0, 100, 70, 40, 10, 80, 50, 20, 90, 60, 30}
```

3. 有 3 个夏令营集合分别如下：(10-2 节)

Math：Peter, Norton, Kevin, Mary, John, Ford, Nelson, Damon, Ivan, Tom

Computer：Curry, James, Mary, Turisa, Tracy, Judy, Lee, Jarmul, Damon, Ivan

Physics：Eric, Lee, Kevin, Mary, Christy, Josh, Nelson, Kazil, Linda, Tom

请分别列出下列资料。

（1）同时参加 3 个夏令营的名单。

（2）同时参加 Math 和 Computer 夏令营的名单。

（3）同时参加 Math 和 Physics 夏令营的名单。

（4）同时参加 Computer 和 Pyhsics 夏令营的名单。

```
====================== RESTART: D:\Python\ex\ex10_3.py ======================
同时参加3个夏令营名单 :  {'Mary'}
同时参加Math和Computer夏令营名单 :  {'Damon', 'Ivan', 'Mary'}
同时参加Math和Physics夏令营名单 :  {'Mary', 'Nelson', 'Kevin', 'Tom'}
同时参加Computer和Physics夏令营名单 :  {'Lee', 'Mary'}
```

4. 请建立两个列表：(10-2 节)

A：1, 3, 5, … , 99

B：1 至 100 的质数

然后求交集，联集，A – B，B – A，AB 对称差集，BA 对称差集。

```
====================== RESTART: D:\Python\ex\ex10_4.py ======================
联集 :  {1, 2, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31, 33, 35, 3
7, 39, 41, 43, 45, 47, 49, 51, 53, 55, 57, 59, 61, 63, 65, 67, 69, 71, 73, 75, 7
7, 79, 81, 83, 85, 87, 89, 91, 93, 95, 97, 99}
交集 :  {3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71
, 73, 79, 83, 89, 97}
A-B差集 :  {1, 9, 15, 21, 25, 27, 33, 35, 39, 45, 49, 51, 55, 57, 63, 65, 69, 75
, 77, 81, 85, 87, 91, 93, 95, 99}
B-A差集 :  {2}
AB对称差集 :  {1, 2, 9, 15, 21, 25, 27, 33, 35, 39, 45, 49, 51, 55, 57, 63, 65,
69, 75, 77, 81, 85, 87, 91, 93, 95, 99}
```

5. 重新设计 ex9_9.py，差别在于将歌曲列表处理成字典时需要使用集合让程序更有效率，另外，打印列表时需要依照字的出现次数由少到多排列，次数相同排列次序可以不必理会。(10-6 节)

```
===================== RESTART: D:/Python/ex/ex10_5.py =====================
dense : 1
python : 1
silenced : 1
at : 1
peters : 1
```

.....

```
implementation : 2
special : 2
of : 3
although : 3
one : 3
idea : 3
be : 3
never : 3
to : 5
the : 6
better : 8
than : 8
is : 10
```

6. 重新设计 ex10_2.py，改为不建立列表直接建立集合 A 和 B 的方式。(10-3 节)

```
===================== RESTART: D:\Python\ex\ex10_6.py =====================
联集 :  {0, 1, 3, 5, 7, 9, 10, 11, 13, 15, 17, 19, 20, 21, 23, 25, 27, 29, 30, 3
1, 33, 35, 37, 39, 40, 41, 43, 45, 47, 49, 50, 51, 53, 55, 57, 59, 60, 61, 63, 6
5, 67, 69, 70, 71, 73, 75, 77, 79, 80, 81, 83, 85, 87, 89, 90, 91, 93, 95, 97, 9
9, 100}
交集 :  {65, 35, 5, 75, 45, 15, 85, 55, 25, 95}
A-B差集 :  {1, 3, 7, 9, 11, 13, 17, 19, 21, 23, 27, 29, 31, 33, 37, 39, 41, 43,
47, 49, 51, 53, 57, 59, 61, 63, 67, 69, 71, 73, 77, 79, 81, 83, 87, 89, 91, 93,
97, 99}
B-A差集 :  {0, 100, 70, 40, 10, 80, 50, 20, 90, 60, 30}
```

若是将这个习题与 ex10_2.py 相比较，读者可以发现程序简化了很多。

7. 请参考程序实例 ch10_27.py，增加下列鸡尾酒。(10-6 节)

(1) 马颈（Horse's Neck）**：**白兰地（Brandy）、姜汁汽水（Ginger Soda）。

(2) 四海一家（Cosmopolitan）**：**伏特加（Vodka）、甜酒（Sweet Wine）、莱姆汁（Lime Juice）、蔓越莓汁（Cranberry Juice）。

(3) 性感沙滩（Sex on the Beach）**：**伏特加（Vodka）、水蜜桃香甜酒（Peach Liqueur）、柳橙汁（Orange Juice）、蔓越莓汁（Cranberry Juice）。

请执行下列输出。

（1）列出含有 Vodka 的酒。

（2）列出含有 Sweet Wine 的酒。

（3）列出含有 Vodka 和 Cranberry Juice 的酒。

（4）列出含有 Vodka 但是没有 Cranberry Juice 的酒。

```
===================== RESTART: D:\Python\ex\ex10_7.py =====================
含有Vodka的酒 :
Bloody Mary
Cosmopolitan
Sex on the Beach
含有Sweet Wine的酒 :
Blue Hawaiian
Cosmopolitan
含有Vodka和Cranberry Juice的酒 :
Cosmopolitan
Sex on the Beach
含有Vodka但是没有Cranberry Juice的酒 :
Bloody Mary
```

11

第 1 1 章

函数设计

本章摘要

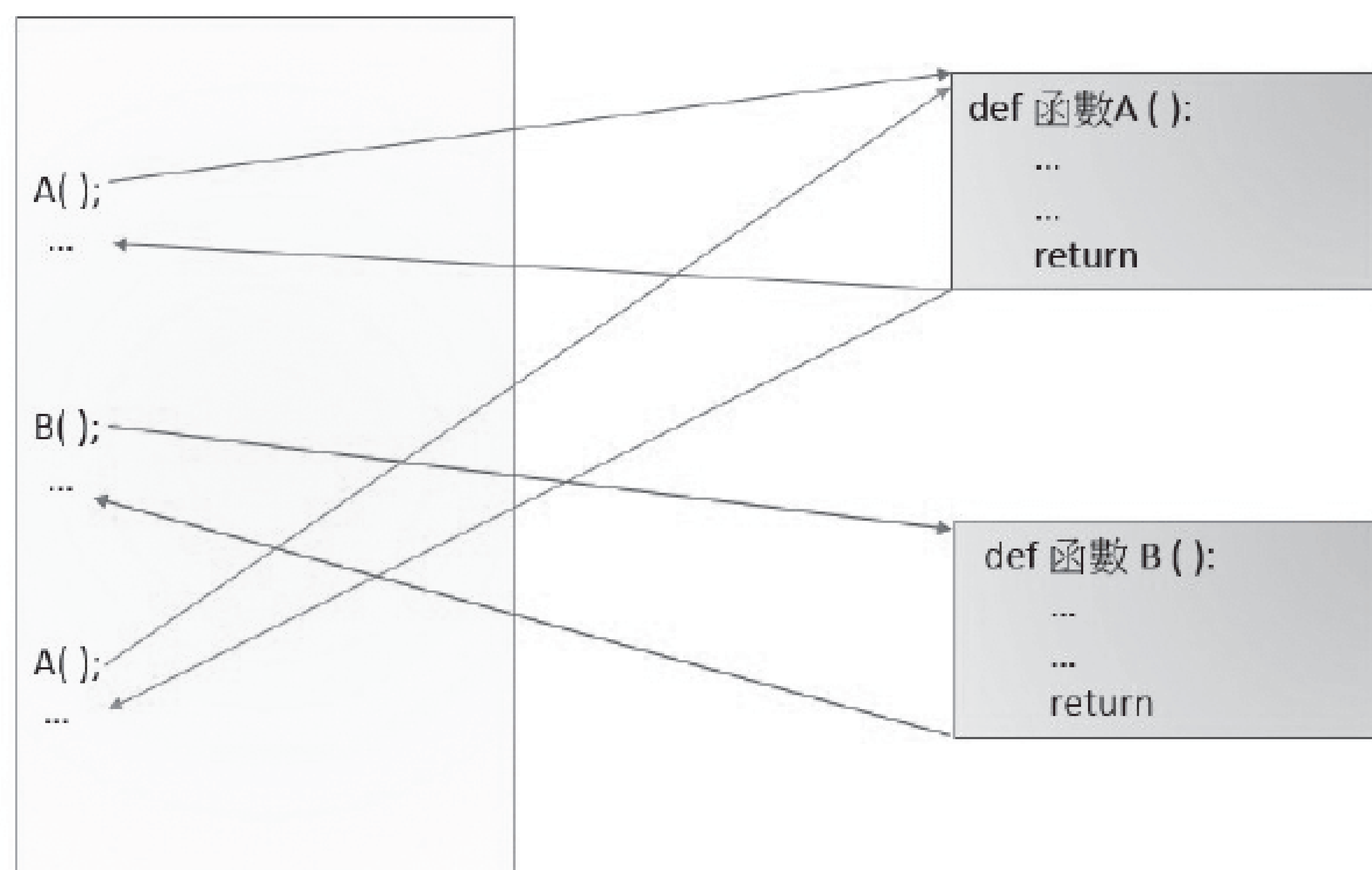
11-1　Python 函数基本概念
11-2　函数的参数设计
11-3　函数返回值
11-4　调用函数时参数是列表
11-5　传递任意数量的参数
11-6　进一步认识函数
11-7　递归式函数设计
11-8　局部变量与全局变量
11-9　匿名函数 lambda
11-10　pass 与函数
11-11　type 关键词应用于函数
11-12　设计自己的 range( )
11-13　装饰器
11-14　专题——函数的应用 / 最大公约数 / 质数

函数（function）其实就是一系列指令语句，它的目的有以下两个。

（1）当我们在设计一个大型程序时，若是能将这个程序依功能将其分割成较小的功能，然后依这些较小的功能要求编写函数程序，如此，不仅使程序简单化，同时最后程序排错也变得容易。另外，编写大型程序时应该是团队合作，每一个人负责一个小功能，以缩短程序开发的时间。

（2）在一个程序中，也许会发生某些指令被重复书写在许多不同的地方，若是能将这些重复的指令编写成一个函数，需要用时再加以调用，如此，不仅减少了编辑程序的时间，同时更可使程序精简、清晰、明了。

下面是调用函数的基本流程图。



当一个程序在调用函数时，Python 会自动跳到被调用的函数上执行工作，执行完后，会回到原先程序的执行位置，然后继续执行下一条指令。

11-1　Python 函数基本概念

通过前面的学习相信读者已经熟悉如何使用 Python 内建的函数了，例如，len( )、add( )、remove( ) 等。有了这些函数，可以随时调用，让程序设计变得很简捷，本章主题是如何设计这类函数。

11-1-1　函数的定义

函数的语法格式如下：

```
def  函数名称 ( 参数值 1[, 参数值 2, … ]):
 """ 函数注释 (docstring)  """
    程序代码区块                              # 需要内缩
    return [ 返回值 1, 返回值 2 , … ]         # 中括号可有可无
```

1. 函数名称

名称必须是唯一的，程序未来可以调用，它的命名规则与一般变量相同。

2. 参数值

可有可无，视函数设计需要，可以接收调用函数传来的变量，各参数值之间用逗号“,”隔开。

3. 函数注释

可有可无，不过如果是参与大型程序设计，当负责一个小程序时，建议给所设计的函数加上注释，除了自己需要也方便他人阅读。注释主要是注明此函数的功能。由于可能是有多行注释所以可以用 3 个双引号（或单引号）括起来。许多英文 Python 资料将此称为 docstring（document string）。

11-6 节将说明如何引用此函数注释。

4. return [返回值 1, 返回值 2 , …]

不论是 return 或接续右边的返回值都是可有可无的，如果有返回多个数据，彼此需以逗号“,”隔开。

11-1-2　没有传入参数也没有返回值的函数

程序实例 ch11_1.py：第一次设计 Python 函数。

```
1  # ch11_1.py
2  def greeting( ):
3      """我的第一个Python函数设计"""
4      print("Python欢迎你")
5      print("祝福学习顺利")
6      print("谢谢")
7
8  # 以下的程序代码也可称主程序
9  greeting( )
10 greeting( )
11 greeting( )
12 greeting( )
13 greeting( )
```

执行结果

```
==================== RESTART: D:\Python\ch11\ch11_1.py ====================
Python欢迎你
祝福学习顺利
谢谢
Python欢迎你
祝福学习顺利
谢谢
Python欢迎你
祝福学习顺利
谢谢
Python欢迎你
祝福学习顺利
谢谢
Python欢迎你
祝福学习顺利
谢谢
```

在程序设计中，有时候也可以将第 8 行以后的程序代码称为**主程序**。读者可以想想看，如果没有函数功能我们的程序设计将如下所示。

程序实例 ch11_2.py：重新设计 ch11_1.py，但是不使用函数设计。

```
1  # ch11_2.py
2  print("Python欢迎你")
3  print("祝福学习顺利")
4  print("谢谢")
5  print("Python欢迎你")
6  print("祝福学习顺利")
7  print("谢谢")
8  print("Python欢迎你")
9  print("祝福学习顺利")
10 print("谢谢")
11 print("Python欢迎你")
12 print("祝福学习顺利")
13 print("谢谢")
14 print("Python欢迎你")
15 print("祝福学习顺利")
16 print("谢谢")
```

执行结果 与 ch11_1.py 相同。

上述程序虽然也可以完成工作，但是可以发现重复的语句太多了，不是一个好的设计。同时如果要将“Python 欢迎你”改成“Python 欢迎你们”，必须修改 5 次相同的语句。经以上讲解读者应该可以了解函数对程序设计的好处了。

11-1-3　在 Python Shell 中执行函数

当程序执行完 ch11_1.py 时，在 Python Shell 窗口中可以看到执行结果，此时也可以在 Python 提示消息（Python prompt）中直接输入 ch11_1.py 程序所建的函数启动与执行。下面是在 Python 提示消息中输入 greeting( ) 函数的实例。

```
==================== RESTART: D:\Python\ch11\ch11_1.py ====================
Python欢迎你
祝福学习顺利
谢谢
Python欢迎你
祝福学习顺利
谢谢
Python欢迎你
祝福学习顺利
谢谢
Python欢迎你
祝福学习顺利
谢谢
Python欢迎你
祝福学习顺利
谢谢
>>> greeting()
Python欢迎你
祝福学习顺利
谢谢
```

11-2　函数的参数设计

11-1 节的程序实例没有传递任何参数，在真实的函数设计与应用中大多是需要传递一些参数的。例如，在前面章节当调用 Python 内建函数时，例如 len( )、print( ) 等，都需要输入参数，接下来将讲解这方面的应用与设计。

11-2-1　传递一个参数

程序实例 ch11_3.py：函数内有参数的应用。

```
1  # ch11_3.py
2  def greeting(name):
3      """Python函数需传递名字name"""
4      print("Hi,", name, "Good Morning!")
5  greeting('Nelson')
```

执行结果

```
==================== RESTART: D:\Python\ch11\ch11_3.py ====================
Hi, Nelson Good Morning!
```

上述语句执行时，第 5 行调用函数 greeting( ) 时，所放的参数是 Nelson，这个字符串将传给函数括号内的 name 参数，所以程序第 4 行会将 Nelson 字符串通过 name 参数打印出来。

在 Python 应用中，有时候也常会将第 4 行写成如下所示，可参考 ch11_3_1.py，执行结果是相同的。

```
4        print("Hi, " + name + " Good Morning!")
```

特别留意由于我们可以在 Python Shell 环境调用函数，所以在设计与使用者（user）交流的程序时，也可以先省略第 5 行的调用，让调用留到 Python 提示消息（prompt）环境。

程序实例 ch11_4.py：程序设计时不做调用，在 Python 提示消息环境再调用。

```
1  # ch11_4.py
2  def greeting(name):
3      """Python函数需传递名字name"""
4      print("Hi, " + name + " Good Morning!")
```

执行结果

```
==================== RESTART: D:\Python\ch11\ch11_4.py ====================
>>> greeting('Nelson')
Hi, Nelson Good Morning!
>>> greeting('Tina')
Hi, Tina Good Morning!
```

上述程序最大的特色是 greeting('Nelson') 与 greeting('Tina')，都是从 Python 提示消息环境做输入。

11-2-2　多个参数传递

当所设计的函数需要传递多个参数时，调用此函数时就需要特别留意传递参数的位置需要正确，最后才可以获得正确的结果。最常见的传递参数是**数值或字符串**数据，在进阶的程序应用中有时也会需要传递**列表、元组、字典或函数**。

程序实例 ch11_5.py：设计减法的函数 subtract( )，第一个参数会减去第二个参数，然后列出执行结果。

```
1  # ch11_5.py
2  def subtract(x1, x2):
3      """ 减法设计 """
4      result = x1 - x2
5      print(result)                  # 输出减法结果
6  print("本程序会执行 a - b 的运算")
7  a = int(input("a = "))
8  b = int(input("b = "))
9  print("a - b = ", end="")          # 输出a-b字符串,接下来输出不换行
10 subtract(a, b)
```

执行结果

```
==================== RESTART: D:\Python\ch11\ch11_5.py ====================
本程序会执行 a - b 的运算
a = 10
b = 5
a - b = 5
```

上述函数的功能是减法运算，所以需要传递两个参数，然后执行第一个数值减去第二个数值。调用这类函数时，就必须留意参数的位置，否则会有错误消息产生。对于上述程序而言，变量 a 和 b 都是从屏幕输入，执行第 9 行调用 subtract( ) 函数时，a 将传给 x1，b 将传给 x2。

程序实例 ch11_6.py：这也是一个需传递两个参数的实例，第一个是兴趣（interest），第二个是主题（subject）。

```
1  # ch11_6.py
2  def interest(interest_type, subject):
3      """ 显示兴趣和主题 """
4      print("我的兴趣是 " + interest_type )
5      print("在 " + interest_type + " 中, 最喜欢的是 " + subject)
6      print( )
7
8  interest('旅游', '敦煌')
9  interest('程序设计', 'Python')
```

执行结果

```
==================== RESTART: D:\Python\ch11\ch11_6.py ====================
我的兴趣是 旅游
在 旅游 中, 最喜欢的是 敦煌

我的兴趣是 程序设计
在 程序设计 中, 最喜欢的是 Python
```

上述程序第 8 行在调用 interest( ) 时，'**旅游**' 会传给 interest_type，'**敦煌**' 会传给 subject。第 9 行在调用 interest( ) 时，'**程序设计**' 会传给 interest_type，'Python' 会传给 subject。对于上述实例，相信读者应该了解调用需要传递多个参数的函数时，所传递参数的位置很重要，否则会有不可预期的错误，如下所示。

```
==================== RESTART: D:\Python\ch11\ch11_6.py ====================
我的兴趣是 旅游
在 旅游 中, 最喜欢的是 敦煌

我的兴趣是 程序设计
在 程序设计 中, 最喜欢的是 Python

>>> interest('敦煌', '旅游')

我的兴趣是 敦煌
在 敦煌 中, 最喜欢的是 旅游
```

11-2-3　关键词参数：参数名称 = 值

关键词参数（keyword arguments）是指调用函数时，参数是用**参数名称 = 值**的配对方式呈现的。Python 也允许在调用需传递多个参数的函数时，直接将**参数名称 = 值**用配对方式传送，这个时候参数的位置就不重要了。

程序实例 ch11_7.py：这个程序基本上是重新设计 ch11_6.py，但是在传递参数时，其中一个参数直接用参数名称 = 值配对方式传送。

```
1  # ch11_7.py
2  def interest(interest_type, subject):
3      """ 显示兴趣和主题 """
4      print("我的兴趣是 " + interest_type )
5      print("在 " + interest_type + " 中, 最喜欢的是 " + subject)
6      print( )
7
8  interest(interest_type = '旅游', subject = '敦煌')   # 位置正确
9  interest(subject = '敦煌', interest_type = '旅游')   # 位置更改
```

执行结果

```
==================== RESTART: D:\Python\ch11\ch11_6.py ====================
我的兴趣是 旅游
在 旅游 中，最喜欢的是 敦煌

我的兴趣是 程序设计
在 程序设计 中，最喜欢的是 Python

>>> interest('敦煌', '旅游')

我的兴趣是 敦煌
在 敦煌 中，最喜欢的是 旅游
```

读者可以留意程序第 8 行和第 9 行的“interest_type = '旅游'”，当调用函数用配对方式传送参数时，即使参数位置不同，程序执行结果也会相同，因为在调用时已经明确指出所传递的值是要给哪一个参数了。

11-2-4　参数默认值的处理

在设计函数时也可以给参数设置**默认值**，如果调用这个函数没有给参数值时，函数的默认值将派上用场。**需特别留意**：函数设计时含有默认值的参数，**必须放置在参数列的最右边**，请参考下列程序第 2 行，如果将“subject = '敦煌'”与“interest_type”位置对调，程序会有错误产生。

程序实例 ch11_8.py：重新设计 ch11_7.py，这个程序会将 subject 的默认值设为“敦煌”。程序将用不同方式调用，读者可以从中体会程序参数默认值的意义。

```
1  # ch11_8.py
2  def interest(interest_type, subject = '敦煌'):
3      """ 显示兴趣和主题 """
4      print("我的兴趣是 " + interest_type )
5      print("在 " + interest_type + " 中，最喜欢的是 " + subject)
6      print( )
7
8  interest('旅游')                                     # 传递一个参数
9  interest(interest_type = '旅游')                     # 传递一个参数
10 interest('旅游', '张家界')                            # 传递二个参数
11 interest(interest_type = '旅游', subject = '张家界')  # 传递二个参数
12 interest(subject = '张家界', interest_type = '旅游')  # 传递二个参数
13 interest('阅读', '旅游类')                            # 传递二个参数,不同的主题
```

执行结果

```
==================== RESTART: D:\Python\ch11\ch11_8.py ====================
我的兴趣是 旅游
在 旅游 中，最喜欢的是 敦煌

我的兴趣是 旅游
在 旅游 中，最喜欢的是 敦煌

我的兴趣是 旅游
在 旅游 中，最喜欢的是 张家界

我的兴趣是 旅游
在 旅游 中，最喜欢的是 张家界

我的兴趣是 旅游
在 旅游 中，最喜欢的是 张家界

我的兴趣是 阅读
在 阅读 中，最喜欢的是 旅游类
```

上述程序第 8 行和 9 行只传递一个参数，所以 subject 就会使用默认值“敦煌”，第 10 行、11 行和 12 行传送了两个参数，其中，第 11 和 12 行用**参数名称 = 值**的配对方式调用传送，可以获得相

同的结果。第 13 行主要说明使用不同类的参数同样可以获得正确的结果。

11-3 函数返回值

在前面的章节实例中有执行调用许多内建的函数，有时会返回一些有意义的数据，例如，len( ) 返回元素数量；有些没有返回值，此时 Python 会自动返回 None，例如 clear( )。为何会如此？本节会完整解说函数返回值的知识。

11-3-1 返回 None

前两节所设计的函数全部没有“return [返回值]”，Python 在直译时会自动返回处理成“return None”，相当于返回 None。在一些程序语言，例如，C 语言中，这个 None 就是 NULL。None 在 Python 中独立成为一个数据类型 NoneType，下面是实例。

程序实例 ch11_9.py：重新设计 ch11_3.py，这个程序并没有做返回值设计，不过笔者将列出 Python 返回 greeting( ) 函数的数据是否是 None，同时列出返回值的数据类型。

```
1  # ch11_9.py
2  def greeting(name):
3      """Python函数需传递名字name"""
4      print("Hi, ", name, " Good Morning!")
5  ret_value = greeting('Nelson')
6  print("greeting( )返回值 = ", ret_value)
7  print(ret_value, " 的 type  = ", type(ret_value))
```

执行结果

```
==================== RESTART: D:\Python\ch11\ch11_9.py ====================
Hi,  Nelson  Good Morning!
greeting( )返回值 =  None
None  的 type  =  <class 'NoneType'>
```

上述函数 greeting( ) 没有 return，Python 将自动处理成 return None。其实即使函数设计时有 return 但是没有返回值，Python 也将自动处理成 return None，可参考下列实例第 5 行。

程序实例 ch11_10.py：重新设计 ch11_9.py，函数末端增加 return。

```
1  # ch11_10.py
2  def greeting(name):
3      """Python函数需传递名字name"""
4      print("Hi, ", name, " Good Morning!")
5      return                  # Python 将自动返回None
6  ret_value = greeting('Nelson')
7  print("greeting( )返回值 = ", ret_value)
8  print(ret_value, " 的 type  = ", type(ret_value))
```

执行结果 与 ch11_9.py 相同。

None 在 Python 中是一个特殊的值，如果将它当作布尔值使用，可将它视为 False，可以参考下列实例。

程序实例 ch11_10_1.py：None 应用于布尔值是 False 的实例。

```
1  # ch11_10_1.py
2  val = None
3  if val:
4      print("I love Java")
5  else:
6      print("I love Python")
```

执行结果

```
==================== RESTART: D:/Python/ch11/ch11_10_1.py ====================
I love Python
```

上述语句由于 val 是 None，可以将其视为 False，所以可以执行第 6 行，输出字符串“I love Python”。其实虽然 None 被视为 False，可是 False 并不是 None。其实空列表、空元组、空字典、空集合虽然是 False，可是它们也不是 None。

上述程序是因教学需要中规中矩的写法，读者容易学习，也可以简化用一行程序代码取代上述 3 ～ 6 行。

程序实例 ch11_10_2.py：高手处理 if … else 的叙述方式。

```
1  # ch11_10_2.py
2  val = None
3  print("I love Java" if val else "I love Python")
```

执行结果 与 ch11_10_1.py 相同。

程序实例 ch11_10_3.py：认识空列表、空元组、空字典、空集合、布尔值 True 与 False 和 None 之间的区别。

```
1   # ch11_10_3.py
2   def is_None(string, x):
3       if x is None:
4           print("%s = None" % string)
5       elif x:
6           print("%s = True" % string)
7       else:
8           print("%s = False" % string)
9
10  is_None("空列表", [])                  # 空列表
11  is_None("空元组", ())                  # 空元组
12  is_None("空字典", {})                  # 空字典
13  is_None("空集合", set())               # 空集合
14  is_None("None  ", None)
15  is_None("True  ", True)
16  is_None("False ", False)
```

执行结果

```
==================== RESTART: D:\Python\ch11\ch11_10_3.py ====================
空列表 = False
空元组 = False
空字典 = False
空集合 = False
None   = None
True   = True
False  = False
```

11-3-2　简单返回数值数据

参数具有返回值的功能，将可以大大增加程序的可读性，返回的基本方式可参考下列程序第 5 行。

```
return result                # result 就是返回的值
```

程序实例 ch11_11.py：利用函数的返回值，重新设计 ch11_5.py 减法的运算。

```
1  # ch11_11.py
2  def subtract(x1, x2):
3      """ 减法设计 """
4      result = x1 - x2
5      return result                   # 返回减法结果
6  print("本程序会执行 a - b 的运算")
7  a = int(input("a = "))
8  b = int(input("b = "))
9  print("a - b = ", subtract(a, b))   # 输出a-b字符串和结果
```

执行结果

```
==================== RESTART: D:\Python\ch11\ch11_11.py ====================
本程序会执行 a - b 的运算
a = 10
b = 5
a - b =  5
```

一个程序常常是由许多函数所组成的，下列是程序含两个函数的应用。

程序实例 ch11_12.py：设计加法和减法器。

```
1  # ch11_12.py
2  def subtract(x1, x2):
3      """ 减法设计 """
4      return x1 - x2                  # 返回减法结果
5  def addition(x1, x2):
6      """ 加法设计 """
7      return x1 + x2                  # 返回加法结果
8
9  # 使用者输入
10 print("请输入运算")
11 print("1:加法")
12 print("2:减法")
13 op = int(input("输入1/2: "))
14 a = int(input("a = "))
15 b = int(input("b = "))
16
17 # 程序运算
18 if op == 1:
19     print("a + b = ", addition(a, b))   # 输出a-b字符串和结果
20 elif op == 2:
21     print("a - b = ", subtract(a, b))   # 输出a-b字符串和结果
22 else:
23     print("运算方法输入错误")
```

执行结果

```
==================== RESTART: D:\Python\ch11\ch11_12.py ====================
请输入运算
1:加法
2:减法
输入1/2: 1
a = 5
b = 3
a + b =  8
>>>
==================== RESTART: D:\Python\ch11\ch11_12.py ====================
请输入运算
1:加法
2:减法
输入1/2: 2
a = 5
b = 3
a - b =  2
```

11-3-3　返回多个数据的应用

使用 return 返回函数数据时，也允许返回多个数据，各个数据间只要以逗号隔开即可，读者可参考下列实例第 8 行。

程序实例 ch11_13.py：请输入两个数据，此函数将返回加法、减法、乘法、除法的执行结果。

```
1  # ch11_13.py
2  def mutifunction(x1, x2):
3      """ 加，减，乘，除四则运算 """
4      addresult = x1 + x2
5      subresult = x1 - x2
6      mulresult = x1 * x2
7      divresult = x1 / x2
8      return addresult, subresult, mulresult, divresult
9
10 x1 = x2 = 10
11 add, sub, mul, div = mutifunction(x1, x2)
12 print("加法结果 = ", add)
13 print("减法结果 = ", sub)
14 print("乘法结果 = ", mul)
15 print("除法结果 = ", div)
```

执行结果

```
===================== RESTART: D:\Python\ch11\ch11_13.py =====================
加法结果 =  20
减法结果 =  0
乘法结果 =  100
除法结果 =  1.0
```

11-3-4　简单返回字符串数据

返回字符串的方法与 11-3-2 节返回数值的方法相同，

程序实例 ch11_14.py：一般中文姓名是 3 个字，笔者将中文姓名拆解为第一个字是姓 lastname，第二个字是中间名 middlename，第三个字是名 firstname。这个程序内有一个函数 guest_info( )，参数意义分别是名 firstname、中间名 middlename 和姓 lastname，以及性别 gender，同时加上问候语返回。

```
1  # ch11_14.py
2  def guest_info(firstname, middlename, lastname, gender):
3      """ 整合客户名字数据 """
4      if gender == "M":
5          welcome = lastname + middlename + firstname + '先生欢迎你'
6      else:
7          welcome = lastname + middlename + firstname + '小姐欢迎妳'
8      return welcome
9
10 info1 = guest_info('宇', '星', '洪', 'M')
11 info2 = guest_info('雨', '冰', '洪', 'F')
12 print(info1)
13 print(info2)
```

执行结果

```
===================== RESTART: D:\Python\ch11\ch11_14.py =====================
洪星宇先生欢迎你
洪冰雨小姐欢迎妳
```

如果是处理外国人的名字，则需在 lastname、middlename 和 firstname 之间加上空格，同时外国人名字处理的顺序是 firstname middlename lastname，这将是读者的习题。

11-3-5　再谈参数默认值

虽然大多数中国人的名字是由 3 个字所组成，但是偶尔也会遇上两个字的状况。其实外国人的名字中，有些人也是只有两个字，因为没有中间名 middlename。如果要让 ch11_14.py 更完美，可以在函数设计时将 middlename 默认为空字符串，这样就可以处理没有中间名的问题。参考 ch11_8.py 可知，设计时必须将默认为空字符串的参数放到函数参数列的最右边。

程序实例 ch11_15.py：重新设计 ch11_14.py，这个程序会将 middlename 默认为空字符串，这样就可以处理没有中间名 middlename 的问题，请留意函数设计时需将此参数放在最右边，可以参考第 2 行。

```
1  # ch11_15.py
2  def guest_info(firstname, lastname, gender, middlename = ''):
3      """ 整合客户名字数据 """
4      if gender == "M":
5          welcome = lastname + middlename + firstname + '先生欢迎你'
6      else:
7          welcome = lastname + middlename + firstname + '小姐欢迎妳'
8      return welcome
9
10 info1 = guest_info('涛', '刘', 'M')
11 info2 = guest_info('雨', '洪', 'F', '冰')
12 print(info1)
13 print(info2)
```

执行结果

```
===================== RESTART: D:\Python\ch11\ch11_15.py =====================
刘涛先生欢迎你
洪冰雨小姐欢迎妳
```

上述第 10 行调用 guest_info( ) 函数时只有 3 个参数，middlename 就会使用默认的空字符串。第 11 行调用 guest_info( ) 函数时有 4 个参数，middlename 就会使用调用函数时所设置的字符串‘冰’。

11-3-6　函数返回字典数据

函数除了可以返回数值或字符串数据外，也可以返回比较复杂的数据，例如，字典或列表等。

程序实例 ch11_16.py：这个程序会调用 build_vip 函数，在调用时会传入 VIP_ID 编号和 Name 姓名数据，函数将返回所建立的字典数据。

```
1  # ch11_16.py
2  def build_vip(id, name):
3      """ 建立VIP信息 """
4      vip_dict = {'VIP_ID':id, 'Name':name}
5      return vip_dict
6
7  member = build_vip('101', 'Nelson')
8  print(member)
```

执行结果

```
==================== RESTART: D:\Python\ch11\ch11_16.py ====================
{'VIP_ID': '101', 'Name': 'Nelson'}
```

上述字典数据只是一个简单的应用，在真正的企业建立 VIP 数据的案例中，可能还需要**性别、电话号码、年龄、电子邮件、地址**等信息。在建立 VIP 数据过程中，也许有些人会愿意提供手机号码，有些人不愿意提供，函数设计时也可以将 Tel 电话号码默认为空字符串，但是如果有电话号码时，程序也可以将它纳入字典内容。

程序实例 ch11_17.py：扩充 ch11_16.py，增加电话号码，调用时若没有电话号码则字典不含此字段，调用时若有电话号码则字典含此字段。

```
1  # ch11_17.py
2  def build_vip(id, name, tel = ''):
3      """ 建立VIP信息 """
4      vip_dict = {'VIP_ID':id, 'Name':name}
5      if tel:
6          vip_dict['Tel'] = tel
7      return vip_dict
8
9  member1 = build_vip('101', 'Nelson')
10 member2 = build_vip('102', 'Henry', '0952222333')
11 print(member1)
12 print(member2)
```

执行结果

```
==================== RESTART: D:\Python\ch11\ch11_17.py ====================
{'VIP_ID': '101', 'Name': 'Nelson'}
{'VIP_ID': '102', 'Name': 'Henry', 'Tel': '0952222333'}
```

程序第 10 行调用 build_vip( ) 函数时，由于有电话号码字段，所以上述程序第 5 行会得到 if 语句的 tel 是 True，所以在第 6 行会将此字段增加到字典中。

11-3-7 将循环应用于建立 VIP 会员字典

我们可以将循环的概念应用于 VIP 会员字典的建立。

程序实例 ch11_18.py：这个程序在执行时基本上是用无限循环的概念，但是当一个数据建立完成时，会询问是否继续，如果输入非 'y' 的字符，程序将执行结束。

```
1  # ch11_18.py
2  def build_vip(id, name, tel = ''):
3      """ 建立VIP信息 """
4      vip_dict = {'VIP_ID':id, 'Name':name}
5      if tel:
6          vip_dict['Tel'] = tel
7      return vip_dict
8
9  while True:
10     print("建立VIP信息系统")
11     idnum = input("请输入ID: ")
12     name = input("请输入姓名: ")
13     tel = input("请输入电话号码: ")        # 如果直接按Enter可不建立此字段
14     member = build_vip(idnum, name, tel)  # 建立字典
15     print(member, '\n')
16     repeat = input("是否继续(y/n)? 输入非y字符可结束系统: ")
17     if repeat != 'y':
18         break
19
20 print("欢迎下次再使用")
```

执行结果

```
===================== RESTART: D:\Python\ch11\ch11_18.py =====================
建立VIP信息系统
请输入ID: 100
请输入姓名: James
请输入电话号码: 0911223344
{'VIP_ID': '100', 'Name': 'James', 'Tel': '0911223344'}

是否继续(y/n)? 输入非y字符可结束系统: y
建立VIP信息系统
请输入ID: 101
请输入姓名: Kevin
请输入电话号码:
{'VIP_ID': '101', 'Name': 'Kevin'}

是否继续(y/n)? 输入非y字符可结束系统: n
欢迎下次再使用
```

笔者在上述输入第 2 个数据时，在电话号码字段没有输入而是直接按 Enter 键，这个动作相当于不做输入，此时将造成可以省略此字段。

11-4 调用函数时参数是列表

11-4-1 基本传递列表参数的应用

在调用函数时，也可以将列表（此列表可以是由**数值、字符串或字典**所组成）当参数传递给函数，然后函数可以遍历列表内容，然后执行更进一步的操作。

程序实例 ch11_19.py：传递列表给 product_msg( ) 函数，函数会遍历列表，然后列出一封产品发表会的信件。

```
1  # ch11_19
2  def product_msg(customers):
3      str1 = '亲爱的: '
4      str2 = '本公司将在2020年12月20日北京举行产品发表会'
5      str3 = '总经理:深石敬上'
6      for customer in customers:
7          msg = str1 + customer + '\n' + str2 + '\n' + str3
8          print(msg, '\n')
9
10 members = ['Damon', 'Peter', 'Mary']
11 product_msg(members)
```

执行结果

```
===================== RESTART: D:\Python\ch11\ch11_19.py =====================
亲爱的: Damon
本公司将在2020年12月20日北京举行产品发表会
总经理:深石敬上

亲爱的: Peter
本公司将在2020年12月20日北京举行产品发表会
总经理:深石敬上

亲爱的: Mary
本公司将在2020年12月20日北京举行产品发表会
总经理:深石敬上
```

11-4-2 观察传递一般变量与列表变量到函数的区别

在讲解修改列表内容前，本节先用两个简单的程序说明传递整数变量与传递列表变量到函数的差别。如果传递的是一般整数变量，其实只是将此变量值传给函数，此变量内容在函数更改时原先主程序的变量值不会改变。

程序实例 ch11_19_1.py：主程序调用函数时传递整数变量，这个程序会在主程序以及函数中列出此变量的值与地址的变化。

```
1  # ch11_19_1.py
2  def mydata(n):
3      print("子程序 id(n) = : ", id(n), "\t", n)
4      n = 5
5      print("子程序 id(n) = : ", id(n), "\t", n)
6
7  x = 1
8  print("主程序 id(x) = : ", id(x), "\t", x)
9  mydata(x)
10 print("主程序 id(x) = : ", id(x), "\t", x)
```

执行结果

```
==================== RESTART: D:\Python\ch11\ch11_19_1.py ====================
主程序 id(x) = :  1349175424      1
子程序 id(n) = :  1349175424      1
子程序 id(n) = :  1349175488      5
主程序 id(x) = :  1349175424      1
```

从上述程序可以发现，主程序在调用 mydata( ) 函数时传递了参数 x，在 mydata( ) 函数中将变量设为 n，当第 4 行变量 n 内容更改为 5 时，这个变量在内存的地址也更改了，所以函数 mydata( ) 执行结束时回到主程序，第 10 行可以得到原先主程序的变量 x 仍然是 1。

如果主程序调用函数所传递的是列表变量，其实是将此列表变量的地址参照传给函数，如果在函数中此列表变量地址参照的内容更改时，原先主程序列表变量内容会随着改变。

程序实例 ch11_19_2.py：主程序调用函数时传递列表变量，这个程序会在主程序以及函数中列出此列表变量的值与地址的变化。

```
1  # ch11_19_2.py
2  def mydata(n):
3      print("函  数 id(n) = : ", id(n), "\t", n)
4      n[0] = 5
5      print("函  数 id(n) = : ", id(n), "\t", n)
6
7  x = [1, 2]
8  print("主程序 id(x) = : ", id(x), "\t", x)
9  mydata(x)
10 print("主程序 id(x) = : ", id(x), "\t", x)
```

执行结果

```
==================== RESTART: D:\Python\ch11\ch11_19_2.py ====================
主程序 id(x) = :  43833864        [1, 2]
函  数 id(n) = :  43833864        [1, 2]
函  数 id(n) = :  43833864        [5, 2]
主程序 id(x) = :  43833864        [5, 2]
```

从上述执行结果可以得到，列表变量的地址不论是在主程序或是函数都保持一致，所以第 4 行函数 mydata( ) 内列表内容改变时，函数执行结束回到主程序可以看到主程序列表内容也更改了。

11-4-3　在函数内修改列表的内容

由 11-4-2 节可以知道 Python 允许主程序调用函数时，传递的参数是列表名称，这时在函数内直接修改列表的内容，同时列表经过修改后，主程序的列表也将随着永久性更改结果。

程序实例 ch11_20.py：设计一个麦当劳的点餐系统，顾客在麦当劳点餐时，可以将所点的餐点放入 unserved 列表，服务完成后将已服务餐点放入 served 列表。

```
1  # ch11_20.py
2  def kitchen(unserved, served):
3      """ 将未服务的餐点转为已经服务 """
4      print("厨房处理顾客所点的餐点")
5      while unserved:
6          current_meal = unserved.pop( )
7          # 模拟出餐点过程
8          print("菜单: ", current_meal)
9          # 将已出餐点转入已经服务列表
10         served.append(current_meal)
11
12 def show_unserved_meal(unserved):
13     """ 显示尚未服务的餐点 """
14     print("=== 下列是尚未服务的餐点 ===")
15     if not unserved:
16         print("*** 没有餐点 ***", "\n")
17     for unserved_meal in unserved:
18         print(unserved_meal)
19
20 def show_served_meal(served):
21     """ 显示已经服务的餐点 """
22     print("=== 下列是已经服务的餐点 ===")
23     if not served:
24         print("*** 没有餐点 ***", "\n")
25     for served_meal in served:
26         print(served_meal)
27
28 unserved = ['大麦克', '劲辣鸡腿堡', '麦克鸡块']   # 所点餐点
29 served = []                                    # 已服务餐点
30
31 # 列出餐厅处理前的点餐内容
32 show_unserved_meal(unserved)                   # 列出未服务餐点
33 show_served_meal(served)                       # 列出已服务餐点
34
35 # 餐厅服务过程
36 kitchen(unserved, served)                      # 餐厅处理过程
37 print("\n", "=== 厨房处理结束 ===", "\n")
38
39 # 列出餐厅处理后的点餐内容
40 show_unserved_meal(unserved)                   # 列出未服务餐点
41 show_served_meal(served)                       # 列出已服务餐点
```

执行结果

```
==================== RESTART: D:\Python\ch11\ch11_20.py ====================
=== 下列是尚未服务的餐点 ===
大麦克
劲辣鸡腿堡
麦克鸡块
=== 下列是已经服务的餐点 ===
*** 没有餐点 ***

厨房处理顾客所点的餐点
菜单:  麦克鸡块
菜单:  劲辣鸡腿堡
菜单:  大麦克

 === 厨房处理结束 ===

=== 下列是尚未服务的餐点 ===
*** 没有餐点 ***

=== 下列是已经服务的餐点 ===
麦克鸡块
劲辣鸡腿堡
大麦克
```

这个程序的主程序从第 28 行开始，将所点的餐点放在 unserved 列表，第 29 行将已经处理的餐点放在 served 列表，程序刚开始是设置空列表。为了了解所做的设置，所以第 32 和 33 行是列出尚未服务的餐点和已经服务的餐点。

程序第 36 行是调用 kitchen( ) 函数，这个程序主要是列出餐点，同时将已经处理的餐点从尚未服务列表 unserved，转入已经服务的列表 served。

程序第 40 和 41 行再执行一次列出尚未服务餐点和已经服务餐点，以便验证整个执行过程。

对于上述程序而言，读者可能会好奇，主程序部分与函数部分是使用相同的列表变量 served 与 unserved，所以经过第 36 行调用 kitchen( ) 后造成列表内容的改变，是否设计这类要更改列表内容的程序时，函数与主程序的变量名称一定要相同？答案是否定的。

程序实例 ch11_21.py：重新设计 ch11_20.py，但是主程序的尚未服务列表改为 order_list，已经服务列表改为 served_list，下面只列出主程序内容。

```
28  order_list = ['大麦克', '劲辣鸡腿堡', '麦克鸡块']     # 所点餐点
29  served_list = []                                      # 已服务餐点
30
31  # 列出餐厅处理前的点餐内容
32  show_unserved_meal(order_list)                        # 列出未服务餐点
33  show_served_meal(served_list)                         # 列出已服务餐点
34
35  # 餐厅服务过程
36  kitchen(order_list, served_list)                      # 餐厅处理过程
37  print("\n", "=== 厨房处理结束 ===", "\n")
38
39  # 列出餐厅处理后的点餐内容
40  show_unserved_meal(order_list)                        # 列出未服务餐点
41  show_served_meal(served_list)                         # 列出已服务餐点
```

执行结果 与 ch11_20.py 相同。

得到上述结果最主要的原因是，当传递列表给函数时，即使函数内的列表与主程序列表是不同的名称，但是函数列表 unserved/served 与主程序列表 order_list/served_list 是指向相同的内存位置，所以在函数更改列表内容时主程序列表内容也随着更改。

11-4-4 使用副本传递列表

有时候在设计餐厅系统时，可能想要保存餐点内容，但是经过先前程序设计可以发现，order_list 列表已经变为空列表了，为了避免这样的情形发生，可以在调用 kitchen( ) 函数时传递副本列表，处理方式如下：

```
kitchen(order_list[:], served_list)     # 传递副本列表（可以参考 6-8-3 节）
```

程序实例 ch11_22.py：重新设计 ch11_21.py，但是保留原 order_list 的内容，整个程序主要是在第 36 行，笔者使用副本传递列表，其他只是程序有一些小调整，例如，原先函数 show_unserved_meal( ) 改名为 show_order_meal( )。

```python
# ch11_22.py
def kitchen(unserved, served):
    """ 将所点的餐点转为已经服务 """
    print("厨房处理顾客所点的餐点")
    while unserved:
        current_meal = unserved.pop( )
        # 模拟出餐点过程
        print("菜单: ", current_meal)
        # 将已出餐点转入已经服务列表
        served.append(current_meal)

def show_order_meal(unserved):
    """ 显示所点的餐点 """
    print("=== 下列是所点的餐点 ===")
    if not unserved:
        print("*** 没有餐点 ***", "\n")
    for unserved_meal in unserved:
        print(unserved_meal)

def show_served_meal(served):
    """ 显示已经服务的餐点 """
    print("=== 下列是已经服务的餐点 ===")
    if not served:
        print("*** 没有餐点 ***", "\n")
    for served_meal in served:
        print(served_meal)

order_list = ['大麦克', '劲辣鸡腿堡', '麦克鸡块']    # 所点餐点
served_list = []                                     # 已服务餐点

# 列出餐厅处理前的点餐内容
show_order_meal(order_list)                          # 列出所点的餐点
show_served_meal(served_list)                        # 列出已服务餐点

# 餐厅服务过程
kitchen(order_list[:], served_list)                  # 餐厅处理过程
print("\n", "=== 厨房处理结束 ===", "\n")

# 列出餐厅处理后的点餐内容
show_order_meal(order_list)                          # 列出所点的餐点
show_served_meal(served_list)                        # 列出已服务餐点
```

执行结果

```
==================== RESTART: D:\Python\ch11\ch11_22.py ====================
=== 下列是所点的餐点 ===
大麦克
劲辣鸡腿堡
麦克鸡块
=== 下列是已经服务的餐点 ===
*** 没有餐点 ***

厨房处理顾客所点的餐点
菜单:  麦克鸡块
菜单:  劲辣鸡腿堡
菜单:  大麦克

 === 厨房处理结束 ===

=== 下列是所点的餐点 ===
大麦克
劲辣鸡腿堡
麦克鸡块
=== 下列是已经服务的餐点 ===
麦克鸡块
劲辣鸡腿堡
大麦克
```

由上述执行结果可以发现，原先存储点餐的 order_list 列表经过 kitchen( ) 函数后，此列表的内容没有改变。

11-4-5　传递列表的提醒

函数传递列表时有一点必须留意，在重复调用过程预设列表时会遗留先前调用的内容。

程序实例 ch11_22_1.py：这个 insertChar( ) 函数有两个参数，第一个参数内容可以是任意数据，

第二个参数是空列表 myList，程序预期是每次调用 insertChar( ) 时将第一个参数内容插入第二个空列表内。

```
1  # ch11_22_1.py
2  def insertChar(letter, myList=[]):
3      myList.append(letter)
4      print(myList)
5
6  insertChar('x')
7  insertChar('y')
```

执行结果

```
==================== RESTART: D:/Python/ch11/ch11_22_1.py ====================
['x']
['x', 'y']
```

从上述执行结果发现，第二次调用 insertChar( ) 时，原先第一次所传递的字符 x 仍然存在 myList 列表内。如果想设计这类程序，建议使用 None 取代 [ ]。

程序实例 ch11_22_2.py：将列表参数默认值设为 None，重新设计 ch11_22_1.py。

```
1  # ch11_22_2.py
2  def insertChar(letter, myList=None):
3      if myList == None:
4          myList = []
5      myList.append(letter)
6      print(myList)
7
8  insertChar('x')
9  insertChar('y')
```

执行结果

```
==================== RESTART: D:/Python/ch11/ch11_22_2.py ====================
['x']
['y']
```

上述语句是在函数内用 if 语句判断是否建立空列表。

11-5 传递任意数量的参数

11-5-1 传递处理任意数量的参数

在设计 Python 的函数时，有时候可能会有多个参数传递到这个函数，此时可以用下列方式设计。

程序实例 ch11_23.py：建立一个冰淇淋的配料程序，一般冰淇淋可以在上面加上配料，这个程序在调用制作冰淇淋函数 make_icecream( ) 时，可以传递 0 到多个配料，然后 make_icecream( ) 函数会将配料结果的冰淇淋列出来。

```
1  # ch11_23.py
2  def make_icecream(*toppings):
3      """ 列出制作冰淇淋的配料 """
4      print("这个冰淇淋所加配料如下")
5      for topping in toppings:
6          print("--- ", topping)
7
8  make_icecream('草莓酱')
9  make_icecream('草莓酱', '葡萄干', '巧克力碎片')
```

执行结果

```
==================== RESTART: D:\Python\ch11\ch11_23.py ====================
这个冰淇淋所加配料如下
---  草莓酱
这个冰淇淋所加配料如下
---  草莓酱
---  葡萄干
---  巧克力碎片
```

上述程序最关键的是第 2 行 make_icecream( ) 函数的参数“*toppings”，这个加上“*”符号的参数代表可以有 0 到多个参数将传递到这个函数内。参数“*toppings”的另一个特点是，它可以将所传递的参数群组化成元组（tuple）。

程序实例 ch11_23_1.py：重新设计 ch11_23.py，验证“*toppings”参数的数据类型是元组。

```
1  # ch11_23_1.py
2  def make_icecream(*toppings):
3      """ 列出制作冰淇淋的配料 """
4      print("这个冰淇淋所加配料如下")
5      for topping in toppings:
6          print("--- ", topping)
7      print(type(toppings))
8      print(toppings)
9
10 make_icecream('草莓酱')
11 make_icecream('草莓酱', '葡萄干', '巧克力碎片')
```

执行结果

```
==================== RESTART: D:\Python\ch11\ch11_23_1.py ====================
这个冰淇淋所加配料如下
---  草莓酱
<class 'tuple'>
('草莓酱',)
这个冰淇淋所加配料如下
---  草莓酱
---  葡萄干
---  巧克力碎片
<class 'tuple'>
('草莓酱', '葡萄干', '巧克力碎片')
```

上述程序第 7 行可以打印 toppings 的数据类型是 <class 'tuple'>，第 8 行可以列出 toppings 的数据内容。上述程序如果调用 make_icecream( ) 时没有传递参数，第 5、6 行的 for 循环将不会执行第 6 行的循环内容。

程序实例 ch11_23_2.py：在调用 make_icecream( ) 时没有传递参数的观察。

```
1  # ch11_23_2.py
2  def make_icecream(*toppings):
3      """ 列出制作冰淇淋的配料 """
4      print("这个冰淇淋所加配料如下")
5      for topping in toppings:
6          print("--- ", topping)
7
8  make_icecream()
```

执行结果

```
==================== RESTART: D:\Python\ch11\ch11_23_2.py ====================
这个冰淇淋所加配料如下
```

11-5-2　设计含有一般参数与任意数量参数的函数

程序设计时有时会遇上需要传递一般参数与任意数量参数，碰上这类状况，任意数量的参数必须放在最右边。

程序实例 ch11_24.py：重新设计 ch11_23.py，传递参数时第一个参数是冰淇淋的种类，然后才是不同数量的冰淇淋的配料。

```
1  # ch11_24.py
2  def make_icecream(icecream_type, *toppings):
3      """ 列出制作冰淇淋的配料 """
4      print("这个 ", icecream_type, " 冰淇淋所加配料如下")
5      for topping in toppings:
6          print("--- ", topping)
7
8  make_icecream('香草', '草莓酱')
9  make_icecream('芒果', '草莓酱', '葡萄干', '巧克力碎片')
```

执行结果

```
===================== RESTART: D:\Python\ch11\ch11_24.py =====================
这个  香草  冰淇淋所加配料如下
---  草莓酱
这个  芒果  冰淇淋所加配料如下
---  草莓酱
---  葡萄干
---  巧克力碎片
```

11-5-3　设计含有一般参数与任意数量的关键词参数

在 11-2-3 节有介绍调用函数的参数是关键词参数（参数是用参数名称 = 值配对方式呈现的），其实也可以设计含任意数量关键词参数的函数，方法是在函数内使用 **kwargs（kWargs 是程序设计师可以自行命名的参数，可以想成 key word arguments），这时关键词参数将会变成任意数量的字典元素，其中，自变量是键，对应的值是字典的值。

程序实例 ch11_25.py：这个程序基本上是用 build_dict( ) 函数建立一个球员的字典数据，主程序会传入一般参数与任意数量的关键词参数，最后可以列出执行结果。

```
1  # ch11_25.py
2  def build_dict(name, age, **players):
3      """ 建立NBA球员的字典数据 """
4      info = {}              # 建立空字典
5      info['Name'] = name
6      info['Age'] = age
7      for key, value in players.items( ):
8          info[key] = value
9      return info            # 返回所建的字典
10
11 player_dict = build_dict('James', '32',
12                          City = 'Cleveland',
13                          State = 'Ohio')
14
15 print(player_dict)         # 打印所建字典
```

执行结果

```
===================== RESTART: D:/Python/ch11/ch11_25.py =====================
{'Name': 'James', 'Age': '32', 'City': 'Cleveland', 'State': 'Ohio'}
>>>
```

上述语句最关键的是第 2 行 build_dict( ) 函数内的参数“**player”，这是可以接受任意数量关键词的参数，它可以将所传递的关键词参数群组化成字典（dict）。

11-6　进一步认识函数

在 Python 中所有东西都是对象，例如，字符串、列表、字典甚至函数也是对象，可以将函数赋值给一个变量，也可以将函数当作参数传送，甚至将函数返回，当然也可以动态建立或是销毁。这让 Python 使用起来非常有弹性，也可以完成其他程序语言无法做到的事情，但是其实也多了一些理解上的难度。

11-6-1　函数文件字符串 docstring

请再看一次 ch11_3.py 程序：

```
1  # ch11_3.py
2  def greeting(name):
3      """Python函数需传递名字name"""
4      print("Hi,", name, "Good Morning!")
5  greeting('Nelson')
```

上述函数 greeting( ) 名称下方是 """Python 函数需 … """ 字符串，Python 语言将此**函数注释**称为**文件字符串** docstring（document string 的缩写）。一个公司在设计大型程序时，常常将工作分成很多小程序，每个人的工作将用函数完成，为了要让其他团队成员了解你所设计的函数，必须用**文件字符串**注明此函数的功能与用法。

可以使用 help（函数名称）列出此函数的文件字符串，参考下列实例。假设已经执行了 ch11_3.py 程序，下面是列出此程序的 greeting( ) 函数的**文件字符串**。

```
>>> help(greeting)
Help on function greeting in module __main__:

greeting(name)
    Python函数需传递名字name
```

如果只是想要看函数注释，可以使用下列方式。

```
>>> print(greeting.__doc__)
Python函数需传递名字name
```

上述语句中奇怪的 greeting.__doc__ 就是 greeting( ) 函数文件字符串的变量名称，“__” 其实是两个下画线，这是系统保留名称的方法，以后会介绍这方面的知识。

11-6-2 函数是一个对象

其实在 Python 中函数也是一个对象，假设有一个函数如下 :

```
>>> def upperStr(text):
        return text.upper()

>>> upperStr('deepstone')
'DEEPSTONE'
```

可以使用对象赋值方式处理此对象，或者说将函数设置给一个变量。

```
>>> upperLetter = upperStr
```

经上述语句执行后 upperLetter 也变成了一个函数，所以可以执行下列操作。

```
>>> upperLetter('deepstone')
'DEEPSTONE'
```

从上述语句执行可以知道，upperStr 和 upperLetter 指的是同一个函数对象。此外，一个函数若是拿掉小括号 ( )，这个函数就是一个内存内的地址了，可参考下列验证。由于 upperStr 和 upperLetter 是指相同对象，所以它们的内存地址相同。

```
>>> upperStr
<function upperStr at 0x0040F150>
>>> upperLetter
<function upperStr at 0x0040F150>
```

如果用 type( ) 观察，可以得到 upperStr 和 upperLetter 都是函数对象。

```
>>> type(upperStr)
<class 'function'>
>>> type(upperLetter)
<class 'function'>
```

11-6-3 函数可以是数据结构成员

函数既然可以是一个对象，就可以将函数当作数据结构（例如，列表、元组 … ）的元素，自然也可以迭代这些函数，这个概念可以应用于自建函数或内建函数。

程序实例 ch11_25_1.py：将所定义的函数 total 与 Python 内建的函数 min( )、max( )、sum( ) 等，当作列表的元素，然后迭代，内建函数会列出 <built-in …>，非内建函数则列出内存地址。

```
1  # ch11_25_1.py
2  def total(data):
3      return sum(data)
4
5  x = (1,5,10)
6  myList = [min, max, sum, total]
7  for f in myList:
8      print(f)
```

执行结果

```
==================== RESTART: D:/Python/ch11/ch11_25_1.py ====================
<built-in function min>
<built-in function max>
<built-in function sum>
<function total at 0x00A9C618>
```

程序实例 ch11_25_2.py：用 for 循环迭代列表内的元素，这些元素是函数，这次有传递参数 (1, 5, 10)。

```
1  # ch11_25_2.py
2  def total(data):
3      return sum(data)
4
5  x = (1,5,10)
6  myList = [min, max, sum, total]
7  for f in myList:
8      print(f, f(x))
```

执行结果

```
==================== RESTART: D:\Python\ch11\ch11_25_2.py ====================
<built-in function min> 1
<built-in function max> 10
<built-in function sum> 16
<function total at 0x04155BB8> 16
```

11-6-4　函数可以当作参数传递给其他函数

在 Python 中函数也可以当作参数传递给其他函数，当函数当作参数传递时，可以不用加上 ( ) 符号，这样 Python 就可以将函数当作对象处理。如果加上括号，会被视为调用这个函数。

程序实例 ch11_25_3.py：函数当作是传递参数的基本应用。

```
1   # ch11_25_3.py
2   def add(x, y):
3       return x+y
4
5   def mul(x, y):
6       return x*y
7
8   def running(func, arg1, arg2):
9       return func(arg1, arg2)
10
11  result1 = running(add, 5, 10)
12  print(result1)
13  result2 = running(mul, 5, 10)
14  print(result2)
```

执行结果

```
==================== RESTART: D:/Python/ch11/ch11_25_3.py ====================
15
50
```

上述第 8 行 running( ) 函数的第 1 个参数是函数，第 2、3 个参数是一般数值，这个 running

函数会依所传递的第一个参数，才会知道要调用 add( ) 或 mul( )，然后才将 arg1 和 arg2 传递给指定的函数。在上述程序中，running( ) 函数可以接受其他函数当作参数的函数，又称其为**高阶函数**（Higher-order function）。

11-6-5　函数当作参数与 *args 不定量的参数

前面已经介绍可以将函数当作传递参数使用，其实也可以配合 *args 与 **kwargs 共同使用。

程序实例 ch11_25_4.py：函数当作参数与 *args 不定量参数配合使用。

```
1  # ch11_25_4.py
2  def mysum(*args):
3      return sum(args)
4
5  def run_with_multiple_args(func, *args):
6      return func(*args)
7
8  print(run_with_multiple_args(mysum,1,2,3,4,5))
9  print(run_with_multiple_args(mysum,6,7,8,9))
```

执行结果

```
==================== RESTART: D:/Python/ch11/ch11_25_4.py ====================
15
30
```

第 5 行 run_with_multiple_args( ) 函数可以接受一个函数与一系列的参数。

11-6-6　嵌套函数

嵌套函数是指函数内部也可以有函数，有时候可以利用这个特性执行复杂的运算。嵌套函数具有可重复使用、封装，隐藏数据的效果。

程序实例 ch11_25_5.py：计算两个坐标点的距离，外层函数是第 2 ～ 7 行的 dist( )，此函数第 3、4 行是内层 mySqrt( ) 函数。

```
1  # ch11_25_5.py
2  def dist(x1,y1,x2,y2):          # 计算两点的距离函数
3      def mySqrt(z):              # 计算开根号值
4          return z ** 0.5
5      dx = (x1 - x2) ** 2
6      dy = (y1 - y2) ** 2
7      return mySqrt(dx+dy)
8
9  print(dist(0,0,1,1))
```

执行结果

```
==================== RESTART: D:/Python/ch11/ch11_25_5.py ====================
1.4142135623730951
```

11-6-7　函数也可以当作返回值

在嵌套函数的应用中，常常会应用到将一个内层函数当作返回值，这时所返回的是内层函数的

内存地址。

程序实例 ch11_25_6.py：计算 1-(n-1) 的总和，观察函数当作返回值的应用，这个程序的第 2 ～ 6 行是 outer( ) 函数，第 6 行的返回值是不含 ( ) 的 inner。

```
1  # ch11_25_6.py
2  def outer():
3      def inner(n):
4          print('inner running')
5          return sum(range(n))
6      return inner
7
8  f = outer()           # outer()返回inner地址
9  print(f)              # 打印inner内存
10 print(f(5))           # 实际执行的是inner()
11
12 y = outer()
13 print(y)
14 print(y(10))
```

执行结果

```
==================== RESTART: D:\Python\ch11\ch11_25_6.py ====================
<function outer.<locals>.inner at 0x02DDF150>
inner running
10
<function outer.<locals>.inner at 0x03201738>
inner running
45
```

这个程序在执行第 8 行时，outer( ) 会返回 inner 的内存地址，所以对于 f 而言所获得的只是内层函数 inner( ) 的内存地址，所以第 9 行可以列出 inner( ) 的内存地址。当执行第 10 行 f(5) 时，才是真正执行计算总和。

由于 inner( ) 是在执行期间被定义，所以第 12 行时会产生新的 inner( ) 地址，所以主程序两次调用会有不同的 inner( )。最后读者必须了解，我们无法在主程序中直接调用内部函数，这会产生错误。

11-6-8　闭包 closure

内部函数是一个动态产生的程序，当它可以记住函数以外的程序所建立的环境变量值时，可以称这个内部函数是**闭包**（closure）。

程序实例 ch11_25_7.py：一个线性函数 ax+b 的闭包说明。

```
1  # ch11_25_7.py
2  def outer():
3      b = 10                    # inner所使用的变量值
4      def inner(x):
5          return 5 * x + b      # 引用第3行的b
6      return inner
7
8  b = 2
9  f = outer()
10 print(f(b))
```

执行结果

```
==================== RESTART: D:/Python/ch11/ch11_25_7.py ====================
20
```

上述语句第 3 行中 b 是一个环境变量，这也是定义在 inner( ) 以外的变量，由于第 6 行使用 inner 当作返回值，inner( ) 内的 b 其实就是第 3 行所定义的 b，变量 b 和 inner( ) 就构成了一个 closure。 程序第 10 行中的 f(b)，其实这个 b 将是 Inner(x) 的 x 参数，所以最后可以得到 5×2 + 10，结果是 20。

其实 __closure__ 内是一个元组，环境变量 b 就是存在 cell_contents 内。

```
>>> print(f)
<function outer.<locals>.inner at 0x0357F150>
>>> print(f.__closure__)
(<cell at 0x039D72D0: int object at 0x5B8EC910>,)
>>> print(f.__closure__[0].cell_contents)
10
```

程序实例 ch11_25_8.py：闭包 closure 的另一个应用，这也是线性函数 ax+b，不过环境变量是 outer( ) 的参数。

```
1  # ch11_25_8.py
2  def outer(a, b):
3      ''' a 和 b 将是inner()的环境变量 '''
4      def inner(x):
5          return a * x + b
6      return inner
7
8  f1 = outer(1, 2)
9  f2 = outer(3, 4)
10 print(f1(1), f2(3))
```

执行结果

```
==================== RESTART: D:/Python/ch11/ch11_25_8.py ====================
3 13
```

这个程序第 8 行相当于建立了 x+2，第 9 行建立了 3x+4，相当于使用了 closure 将最终线性函数确定下来，第 10 行传递适当的值，就可以获得结果。在这里我们发现程序代码可以重复使用，此外，如果没有 closure，我们需要传递 a、b、x 参数，所以 closure 可以让程序设计更有效率，同时以后扩充时程序代码更容易移植。

11-7 递归式函数设计

一个函数可以调用其他函数，也可以调用自己，其中，调用本身的动作称为**递归式**（recursive）调用，递归式调用有下列特点。

（1）每次调用自己时，都会使范围越来越小。

（2）必须要有一个终止的条件来结束递归函数。

递归函数可以使程序变得很简洁，但是设计这类程序时如果不小心很容易进入无限循环的陷阱，所以使用这类函数时一定要特别小心。递归函数最常见的应用是处理正整数的**阶乘**（factorial），一个正整数的阶乘是所有小于以及等于该数的正整数的积，同时如果正整数是 0 则阶乘为 1，依照概念正整数是 1 时阶乘也是 1。此阶乘数字的表示法为 n!。

实例 1：n 是 3，下列是阶乘数的计算方式。

n! = 1×2×3

结果是 6

实例 2：n 是 5，下列是阶乘数的计算方式。

n! = 1×2×3×4×5

结果是 120

阶乘数的概念是由法国数学家克里斯蒂安 • 克兰普（Christian Kramp, 1760—1826）所发表，他虽然学医但是却同时对数学感兴趣，发表了许多数学文章。

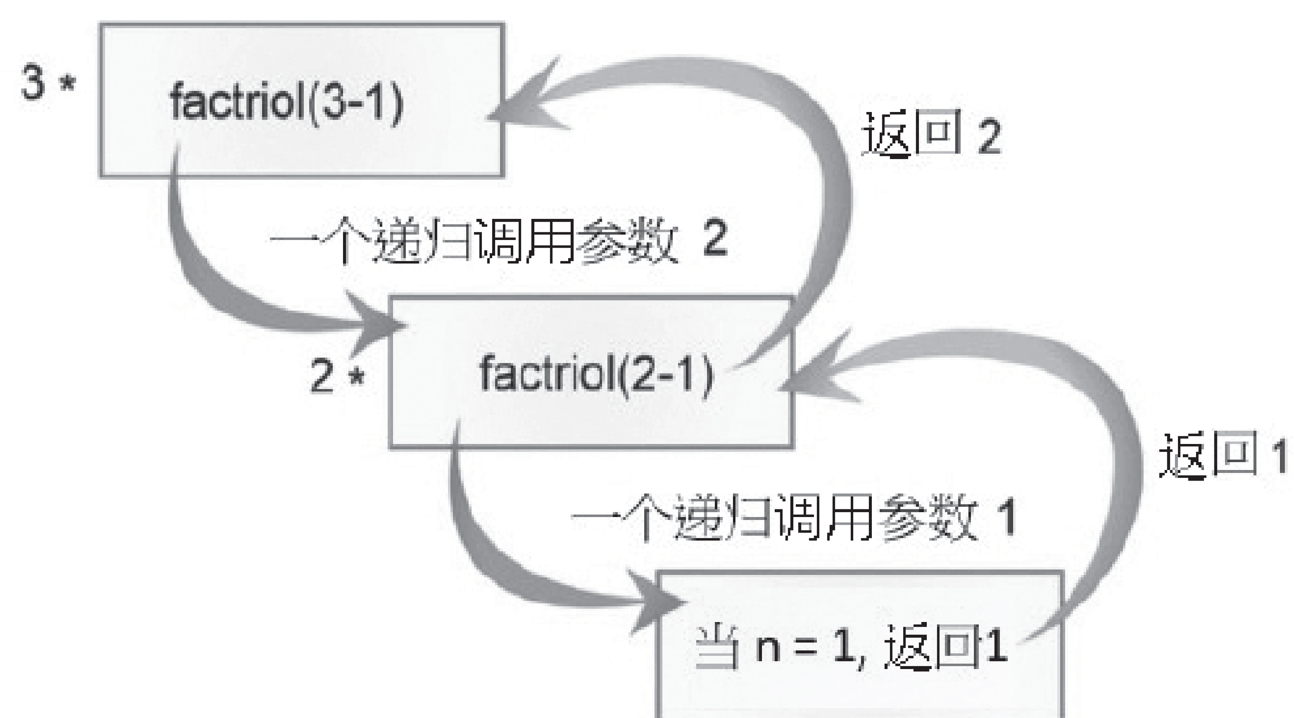
程序实例 ch11_26.py：使用递归函数执行阶乘（factorial）运算。

```
1  # ch11_26.py
2  def factorial(n):
3      """ 计算n的阶乘, n 必须是正整数 """
4      if n == 1:
5          return 1
6      else:
7          return (n * factorial(n-1))
8
9  value = 3
10 print(value, " 的阶乘结果是 = ", factorial(value))
11 value = 5
12 print(value, " 的阶乘结果是 = ", factorial(value))
```

执行结果

```
==================== RESTART: D:\Python\ch11\ch11_26.py ====================
3  的阶乘结果是 =  6
5  的阶乘结果是 =  120
```

上述 factorial( ) 函数的终止条件是参数值为 1 的情况，由第 4 行判断然后返回 1，下列是正整数为 3 时递归函数的情况讲解。



Python 预设最大递归次数为 1000 次，可以先导入 sys 模块，第 13 章会介绍导入模块的更多知识。读者可以使用 sys.getrecursionlimit( ) 列出 Python 预设或目前递归的最大次数。

```
>>> import sys
>>> sys.getrecursionlimit()
1000
```

sys.setrecursionlimit( ) 可以设置最大递归次数。

11-8 局部变量与全局变量

在设计函数时，另一个重点是适当地使用变量名称。某个变量只能在该函数内使用，影响范围限定在这个函数内，这个变量称为**局部变量**（local variable）。如果某个变量的影响范围是整个程

序，则这个变量称为**全局变量**（global variable）。

Python 程序在调用函数时会建立一个内存工作区间，在这个内存工作区间可以处理属于这个函数的变量，当函数工作结束，返回原先的调用程序时，这个内存工作区间就被收回，原先存在的变量也将被销毁，这也是为何**局部变量**的影响范围只限定在所属的函数内。

对于**全局变量**而言，一般是在主程序内建立，程序在执行时，不仅主程序可以引用，所有属于这个程序的函数也可以引用，所以它的影响范围是整个程序。

11-8-1 全局变量可以在所有函数中使用

一般在主程序内建立的变量称为全局变量，这个变量可以供主程序内与本程序的所有函数引用。

程序实例 ch11_27.py：这个程序会设置一个全局变量，然后函数也可以调用。

```
1  # ch11_27.py
2  def printmsg():
3      """ 函数本身没有定义变量，只有执行打印全局变量功能 """
4      print("函数打印: ", msg)    # 打印全局变量
5
6  msg = 'Global Variable'        # 设置全局变量
7  print("主程序行印: ", msg)      # 打印全局变量
8  printmsg()                     # 调用函数
```

执行结果

```
==================== RESTART: D:\Python\ch11\ch11_27.py ====================
主程序行印:  Global Variable
函数打印:  Global Variable
```

11-8-2 局部变量与全局变量使用相同的名称

在设计程序时建议对全局变量与函数内的局部变量不要使用相同的名称，因为对新手而言很容易造成混淆。如果发生全局变量与函数内的局部变量使用相同的名称时，Python 会将相同名称的区域与全局变量视为不同的变量，在局部变量所在的函数中是使用局部变量内容，其他区域则是使用全局变量的内容。

程序实例 ch11_28.py：局部变量与全局变量定义了相同的变量 msg，但是内容不相同。然后执行打印，可以发现在函数与主程序中所打印的内容有不同的结果。

```
1  # ch11_28.py
2  def printmsg():
3      """ 函数本身有定义变量，将执行打印局部变量功能 """
4      msg = 'Local Variable'     # 设置局部变量
5      print("函数打印: ", msg)    # 打印局部变量
6
7  msg = 'Global Variable'        # 这是全局变量
8  print("主程序行印: ", msg)      # 打印全局变量
9  printmsg()                     # 调用函数
```

执行结果

```
==================== RESTART: D:\Python\ch11\ch11_28.py ====================
主程序行印:  Global Variable
函数打印:  Local Variable
```

11-8-3　程序设计注意事项

一般程序设计时在使用局部变量时需注意下列事项，否则程序会有错误产生。

（1）局部变量内容无法在其他函数引用，可参考 ch11_29.py。

（2）局部变量内容无法在主程序引用，可参考 ch11_30.py。

（3）在函数内不能更改全局变量的值，可参考 ch11_30_1.py。

（4）如果要在函数内存取或修改全局变量值，需在函数内使用 global 声明此变量，可参考 ch11_30_2.py。

程序实例 ch11_29.py：局部变量在其他函数中引用，造成程序错误的应用。

```
1  # ch11_29.py
2  def defmsg():
3      msg = 'pringmsg variable'
4
5  def printmsg():
6      print(msg)        # 打印defmsg()函数定义的局部变量
7
8  printmsg()            # 调用printmsg()
```

执行结果

```
===================== RESTART: D:\Python\ch11\ch11_29.py =====================
Traceback (most recent call last):
  File "D:\Python\ch11\ch11_29.py", line 8, in <module>
    printmsg()          # 呼叫printmsg()
  File "D:\Python\ch11\ch11_29.py", line 6, in printmsg
    print(msg)      # 打印defmsg()函数定义的局部变量
NameError: name 'msg' is not defined
```

上述程序的错误原因主要是 printmsg( ) 函数内没有定义 msg 变量，所以产生程序错误。

程序实例 ch11_30.py：局部变量在主程序中引用产生错误的实例。

```
1  # ch11_30.py
2  def defmsg():
3      msg = 'pringmsg variable'
4
5  print(msg)          # 主程序行打印局部变量产生错误
```

执行结果

```
===================== RESTART: D:\Python\ch11\ch11_30.py =====================
Traceback (most recent call last):
  File "D:\Python\ch11\ch11_30.py", line 5, in <module>
    print(msg)          # 主程序行打印局部变量产生错误
NameError: name 'msg' is not defined
```

上述程序的错误原因主要是主程序内没有定义 msg 变量，所以产生程序错误。

程序实例 ch11_30_1.py：在函数内尝试更改全局变量，结果是增加定义一个局部变量。

```
1  # ch11_30_1.py
2  def printmsg():
3      msg = "Java"          # 尝试更改全局变量造成建立一个局部变量
4      print("更改后: ", msg)
5  msg = "Python"
6  printmsg()
```

执行结果

```
==================== RESTART: D:\Python\ch11\ch11_30_1.py ====================
更改后:  Java
```

如果全局变量在函数内可能更改内容时，需要在函数内使用 global 声明这个全局变量，程序才不会有错。

程序实例 ch11_30_2.py：使用 global 在函数内声明全局变量。

```
1  # ch11_30_2.py
2  def printmsg():
3      global msg
4      msg = "Java"          # 更改全局变量
5      print("更改后: ", msg)
6  msg = "Python"
7  print("更改前: ", msg)
8  printmsg()
```

执行结果

```
==================== RESTART: D:\Python\ch11\ch11_30_2.py ====================
更改前:  Python
更改后:  Java
```

11-8-4 locals( ) 和 globals( )

Python 提供函数让我们了解目前变量的名称与内容。

locals( )：用字典方式列出所有的局部变量名称与内容。

globals( )：用字典方式列出所有的全局变量名称与内容。

程序实例 ch11_30_3.py：列出所有局部变量与全局变量的内容。

```
1  # ch11_30_3.py
2  def printlocal():
3      lang = "Java"
4      print("语言 : ", lang)
5      print("局部变量 : ", locals())
6  msg = "Python"
7  printlocal()
8  print("语言 : ", msg)
9  print("全局变量 : ",globals())
```

执行结果

```
==================== RESTART: D:\Python\ch11\ch11_30_3.py ====================
语言 :  Java
局部变量 :  {'lang': 'Java'}
语言 :  Python
全局变量 :  {'__name__': '__main__', '__doc__': None, '__package__': None, '__lo
ader__': <class '_frozen_importlib.BuiltinImporter'>, '__spec__': None, '__annot
ations__': {}, '__builtins__': <module 'builtins' (built-in)>, '__file__': 'D:\\
Python\\ch11\\ch11_30_3.py', 'printlocal': <function printlocal at 0x036A0420>,
'msg': 'Python'}
```

请留意在上述全局变量中，除了最后一个 'msg':'Python' 是程序设置的，其他均是系统内建，后面会针对此部分做说明。

11-9 匿名函数 lambda

匿名函数（anonymous function）是指一个没有名称的函数，适合在程序中只存在一小段时间的情况。Python 使用 def 定义一般函数，匿名函数则是使用 lambda 来定义，有人称之为 lambda 表达式，也可以将匿名函数称为 lambda 函数。有时会将匿名函数与 Python 的内建函数 filter( )、map( )、reduce( ) 等共同使用，此时匿名函数将只是这些函数的参数，后面将以实例做进行讲解。

11-9-1 匿名函数 lambda 的语法

匿名函数最大的特点是可以有许多的参数，但是只能有一个表达式，然后可以将执行结果返回。

```
lambda arg1[, arg2, …,argn]:expression    # arg1 是参数，可以有多个参数
```

其中，expression 就是匿名函数 lambda 表达式的内容。

程序实例 ch11_31.py：使用一般函数设计返回平方值。

```
1  # ch11_31.py
2  # 使用一般函数
3  def square(x):
4      value = x ** 2
5      return value
6
7  # 输出平方值
8  print(square(10))
```

执行结果

```
==================== RESTART: D:/Python/ch11/ch11_31.py ====================
100
```

程序实例 ch11_32.py：单一参数的匿名函数应用，可以返回平方值。

```
1  # ch11_32.py
2  # 定义lambda函数
3  square = lambda x: x ** 2
4
5  # 输出平方值
6  print(square(10))
```

执行结果 与 ch11_31.py 相同。

下列是匿名函数含有多个参数的应用。

程序实例 ch11_33.py：含两个参数的匿名函数应用，可以返回参数的积（相乘的结果）。

```
1  # ch11_33.py
2  # 定义lambda函数
3  product = lambda x, y: x * y
4
5  # 输出相乘结果
6  print(product(5, 10))
```

执行结果

```
===================== RESTART: D:\Python\ch11\ch11_33.py =====================
50
```

11-9-2 使用 lambda 匿名函数的时机

使用 lambda 函数的最佳时机是在一个函数的内部，可以参考下列实例。

程序实例 ch11_33_1.py：这是一个 2x+b 方程式，有两个变量，第 5 行定义 linear 时，才确定 lambda 方程式是 2x+5，所以第 6 行可以得到 25。

```
1  # ch11_33_1.py
2  def func(b):
3      return lambda x : 2 * x + b
4
5  linear  = func(5)        # 5将传给lambda的 b
6  print(linear(10))        # 10是lambda的 x
```

执行结果

```
==================== RESTART: D:/Python/ch11/ch11_33_1.py ====================
25
```

程序实例 ch11_33_2.py：重新设计 ch11_33_1.py，使用一个函数但是有两个方程式。

```
1  # ch11_33_2.py
2  def func(b):
3      return lambda x : 2 * x + b
4
5  linear  = func(5)        # 5将传给lambda的 b
6  print(linear(10))        # 10是lambda的 x
7
8  linear2 = func(3)
9  print(linear2(10))
```

执行结果

```
==================== RESTART: D:/Python/ch11/ch11_33_2.py ====================
25
23
```

11-9-3 匿名函数应用于高阶函数的参数

匿名函数一般是用在不需要函数名称的场合，例如，一些高阶函数（Higher-order function）的部分参数是函数，这时就很适合使用匿名函数，同时让程序变得更简洁。在正式以实例讲解前，我们先举一个使用一般函数当作函数参数的实例。

程序实例 ch11_33_3.py：以一般函数当作函数参数的实例。

```
1  # ch11_33_3.py
2  def mycar(cars,func):
3      for car in cars:
4          print(func(car))
5  def wdcar(carbrand):
6      return "My dream car is " + carbrand.title()
7
8  dreamcars = ['porsche','rolls royce','maserati']
9  mycar(dreamcars, wdcar)
```

执行结果

```
==================== RESTART: D:\Python\ch11\ch11_33_3.py ====================
My dream car is Porsche
My dream car is Rolls Royce
My dream car is Maserati
```

上述程序第 9 行调用 mycar( ) 使用两个参数，第 1 个参数是 dreamcars 字符串，第 2 个参数是 wdcar( ) 函数，wdcar( ) 函数的功能是结合字符串“My dream car is”和将 dreamcars 列表元素的字符串第 1 个字母用大写。

其实上述 wdcar( ) 函数就是使用匿名函数的好时机。

程序实例 ch11_33_4.py：重新设计 ch11_33_3.py，使用匿名函数取代 wdcar( )。

```
1  # ch11_33_4.py
2  def mycar(cars,func):
3      for car in cars:
4          print(func(car))
5
6  dreamcars = ['porsche','rolls royce','maserati']
7  mycar(dreamcars, lambda carbrand:"My dream car is " + carbrand.title())
```

执行结果 与 ch11_33_3.py 相同。

18-4-3 节会以实例介绍使用 lambda 表达式的好时机。

11-9-4　匿名函数的使用与 filter( )

有一个内建函数 filter( )，主要是筛选序列，它的语法格式如下：

filter(func, iterable)

上述函数将依次将 iterable（可以重复执行，例如，字符串 string、列表 list 或元组 tuple）的元素（item）放入 func(item) 内，然后将 func( ) 函数执行结果是 True 的元素（item）组成新的筛选对象（filter object）返回。

程序实例 ch11_34.py：使用传统函数定义方式将列表元素内容是奇数的元素筛选出来。

```
1  # ch11_34.py
2  def oddfn(x):
3      return x if (x % 2 == 1) else None
4
5  mylist = [5, 10, 15, 20, 25, 30]
6  filter_object = filter(oddfn, mylist)      # 返回filter object
7
8  # 输出奇数列表
9  print("奇数列表: ",[item for item in filter_object])
```

执行结果

```
==================== RESTART: D:\Python\ch11\ch11_34.py ====================
奇数列表:  [5, 15, 25]
```

第 9 行使用 item for item in filter_object，这是可以取得 filter object 元素的方式，这个操作方式与下列 for 循环类似。

```
for item in filter_object:
 print(item)
```

若是想要获得列表结果，可以使用下列方式。

```
oddlist = [item for item in filter_object]
```

程序实例 ch11_35.py：重新设计 ch11_34.py，将 filter object 转为列表，下面只列出与 ch11_34.py 不同的程序代码。

```
7  oddlist = [item for item in filter_object]
8  # 输出奇数列表
9  print("奇数列表: ",oddlist)
```

执行结 与 ch11_34.py 相同。

匿名函数的最大优点是可以让程序变得更简洁，可参考下列程序实例。

程序实例 ch11_36.py：使用匿名函数重新设计 ch11_35.py。

```
1  # ch11_36.py
2  mylist = [5, 10, 15, 20, 25, 30]
3
4  oddlist = list(filter(lambda x: (x % 2 == 1), mylist))
5
6  # 输出奇数列表
7  print("奇数列表: ",oddlist)
```

执行结果 与 ch11_35.py 相同。

上述程序第 4 行直接使用 list( ) 函数将返回的 filter object 转成列表了。

11-9-5　匿名函数的使用与 map( )

Google 有一篇大数据领域著名的论文 *MapReduce:Simplified Data Processing on Large Clusters*，接下来的两节将介绍 map( ) 和 reduce( ) 函数。

有一个内建函数 map( )，它的语法格式如下：

```
map(func, iterable)
```

上述函数依次将 iterable 重复执行，例如，字符串 string、列表 list 或元组（tuple）的元素（item）放入 func(item) 内，然后将 func( ) 函数执行结果返回。

程序实例 ch11_37.py：使用匿名函数对列表元素执行平方运算。

```
1  # ch11_37.py
2  mylist = [5, 10, 15, 20, 25, 30]
3
4  squarelist = list(map(lambda x: x ** 2, mylist))
5
6  # 输出列表元素的平方值
7  print("列表的平方值: ",squarelist)
```

执行结果

```
====================== RESTART: D:\Python\ch11\ch11_37.py ======================
列表的平方值:  [25, 100, 225, 400, 625, 900]
```

11-9-6　匿名函数的使用与 reduce( )

内建函数 reduce( ) 的语法格式如下：

reduce(func, iterable)　　　　　　# func 必须有两个参数

它会先对可迭代对象的第 1 个和第 2 个元素操作，结果再和第 3 个元素操作，直到最后一个元素。假设 iterable 有 4 个元素，可以用下列方式。

reduce（f, [a, b, c, d]）= f（ f（ f（a, b），c），d）

早期 reduce( ) 是内建函数，现在已被移至 funtools，所以使用时需在程序前方加上 import。

import functools as reduce　　　　　# 导入 reduce( )

程序实例 ch11_37_1.py：设计字符串转整数的函数，为了验证转整数结果正确，将此字符串加 10，最后再输出。

```
1  # ch11_37_1.py
2  from functools import reduce
3  def strToInt(s):
4      def func(x, y):
5          return 10*x+y
6      def charToNum(s):
7          print("s = ", type(s), s)
8          mydict = {'0':0,'1':1,'2':2,'3':3,'4':4,'5':5,'6':6,'7':7,'8':8,'9':9}
9          n = mydict[s]
10         print("n = ", type(n), n)
11         return n
12     return reduce(func,map(charToNum,s))
13
14 string = '5487'
15 x = strToInt(string) + 10
16 print("x = ", x)
```

执行结果

```
==================== RESTART: D:/Python/ch11/ch11_37_1.py ====================
s =  <class 'str'> 5
n =  <class 'int'> 5
s =  <class 'str'> 4
n =  <class 'int'> 4
s =  <class 'str'> 8
n =  <class 'int'> 8
s =  <class 'str'> 7
n =  <class 'int'> 7
x =  5497
```

这本书是以教学为目的，所以会讲解程序演变过程，上述程序第 8 行和第 9 行可以简化如下。

```
8          n = {'0':0,'1':1,'2':2,'3':3,'4':4,'5':5,'6':6,'7':7,'8':8,'9':9}[s]
```

可以参考本书代码文件 ch11_37_2.py，当然也可以进一步简化 charToNum( ) 函数如下。

```
6      def charToNum(s):
7          return {'0':0,'1':1,'2':2,'3':3,'4':4,'5':5,'6':6,'7':7,'8':8,'9':9}[s]
8      return reduce(func,map(charToNum,s))
```

可以参考本书代码文件 ch11_37_3.py。

程序实例 ch11_37_4.py：使用 lambda 简化前一个程序设计。

```
1  # ch11_37_4.py
2  from functools import reduce
3  def strToInt(s):
4      def charToNum(s):
5          return {'0':0,'1':1,'2':2,'3':3,'4':4,'5':5,'6':6,'7':7,'8':8,'9':9}[s]
6      return reduce(lambda x,y:10*x+y, map(charToNum,s))
7
8  string = '5487'
9  x = strToInt(string) + 10
10 print("x = ", x)
```

执行结果 与 ch11_37_1.py 相同。

11-10 pass 与函数

在 7-4-8 节已经有对 pass 指令做过介绍，其实当我们在设计大型程序时，可能会先规划各个函数的功能，然后再逐一完成各个函数设计，但是在程序完成前可以先将尚未完成的函数内容放上 pass。

程序实例 ch11_38.py：将 pass 应用于函数设计。

```
1  # ch11_38.py
2  def fun(arg):
3      pass
```

执行结果 程序没有执行结果。

11-11 type 关键词应用于函数

在结束本章前列出函数的数据类型，读者可以参考。

程序实例 ch11_39.py：输出函数与匿名函数的数据类型。

```
1  # ch11_39.py
2  def fun(arg):
3      pass
4
5  print("列出fun的type类型    :       ", type(fun))
6  print("列出lambda的type类型:        ", type(lambda x:x))
7  print("列出内建函数abs的type类型: ", type(abs))
```

执行结果

```
===================== RESTART: D:\Python\ch11\ch11_39.py =====================
列出fun的type类型    :        <class 'function'>
列出lambda的type类型:        <class 'function'>
列出内建函数abs的type类型:  <class 'builtin_function_or_method'>
```

11-12　设计自己的 range( )

在 Python 2 中，range( ) 所返回的是列表，在 Python 3 版本中所返回的则是 range 对象。range( ) 最大的特点是它不需要预先存储所有序列范围的值，因此可以节省内存与增加程序效率，每次迭代时，它会记得上次调用的位置同时返回下一个位置，这是一般函数做不到的。

程序实例 ch11_39_1.py：设计自己的 range( ) 函数，此函数名称是 myRange( )。

```
1  # ch11_39_1.py
2  def myRange(start=0, stop=100, step=1):
3      n = start
4      while n < stop:
5          yield n
6          n += step
7
8  print(type(myRange))
9  for x in myRange(0,5):
10     print(x)
```

执行结果

```
==================== RESTART: D:/Python/ch11/ch11_39_1.py ====================
<class 'function'>
0
1
2
3
4
```

上述设计的 myRange( ) 函数，数据类型是 function，所执行的功能与 range( ) 类似，不过当调用此函数时，它的返回值不是使用 return，而是使用 yield，同时整个函数内部不是立即执行。第一次 for 循环执行时会执行到 yield 关键词，然后返回 n 值。下一次 for 循环迭代时会继续执行此函数的第 6 行 "n += step"，然后回到函数起点再执行到 yield，循环直到没有值可以返回。

我们又将此 range( ) 称为**生成器**（generator）。

11-13　装饰器

在程序设计时我们会设计一些函数，有时候想在函数内增加一些功能，但是又不想更改原先的函数，这时可以使用 Python 所提供的**装饰器**（decorator）。装饰器其实也是一种函数，基本上此函数会接收一个函数，然会返回另一个函数。下面是一个简单打印所传递的字符串然后输出的实例。

```
>>> def greeting(string):
        return string

>>> greeting('Hello! iPhone')
'Hello! iPhone'
```

假设不想更改 greeting( ) 函数内容，但是希望将输出改成大写，此时就是使用装饰器的时机。

程序实例 ch11_39_2.py：装饰器函数的基本操作。这个程序将设计一个 upper( ) 装饰器，这个程序除了将所输入字符串改成大写，同时也列出所装饰的函数名称，以及函数所传递的参数。

```
1  # ch11_39_2.py
2  def upper(func):                    # 装饰器
3      def newFunc(args):
4          oldresult = func(args)
5          newresult = oldresult.upper()
6          print('函数名称 : ', func.__name__)
7          print('函数参数 : ', args)
8          return newresult
9      return newFunc
10
11 def greeting(string):               # 问候函数
12     return string
13
14 mygreeting = upper(greeting)        # 手动装饰器
15 print(mygreeting('Hello! iPhone'))
```

执行结果

```
==================== RESTART: D:\Python\ch11\ch11_39_2.py ====================
函数名称 :  greeting
函数参数 :  Hello! iPhone
HELLO! IPHONE
```

上述程序第 14 行是手动设置装饰器，第 15 行是调用装饰器和打印。

装饰器设计的原则是有一个函数当作参数，然后在装饰器内重新定义一个含有装饰功能的新函数，可参考第 3 ～ 8 行。第 4 行是获得原函数 greeting( ) 的结果，第 5 行是将 greeting( ) 的结果装饰成新的结果，也就是将字符串转成大写。第 6 行是打印原函数的名称，在这里使用了 func.__name__，这是函数名称变量。第 7 行是打印所传递参数内容，第 8 行是返回新的结果。

上述第 14 行是手动设置装饰器，在 Python 中可以在要装饰的函数前面加上 @decorator，直接定义装饰器。

程序实例 ch11_39_3.py：第 10 行直接使用 @upper 定义装饰器方式，取代手动定义装饰器，重新设计 ch11_39_2.py，程序第 14 行可以直接调用 greeting( ) 函数。

```
1  # ch11_39_3.py
2  def upper(func):                    # 装饰器
3      def newFunc(args):
4          oldresult = func(args)
5          newresult = oldresult.upper()
6          print('函数名称 : ', func.__name__)
7          print('函数参数 : ', args)
8          return newresult
9      return newFunc
10 @upper                              # 设置装饰器
11 def greeting(string):               # 问候函数
12     return string
13
14 print(greeting('Hello! iPhone'))
```

执行结果 与 ch11_39_2.py 相同。

装饰器另一个常用概念是为一个函数增加除错的检查功能，例如，有一个除法函数如下。

```
>>> def mydiv(x,y):
        return x/y

>>> mydiv(6,2)
3.0
>>> mydiv(6,0)
Traceback (most recent call last):
  File "<pyshell#22>", line 1, in <module>
    mydiv(6,0)
  File "<pyshell#20>", line 2, in mydiv
    return x/y
ZeroDivisionError: division by zero
```

很明显若 div( ) 的第 2 个参数是 0 时，将造成除法错误，不过可以使用装饰器修改此除法功能。

程序实例 ch11_39_4.py：设计一个装饰器 @errcheck，为一个除法增加除数为 0 的检查功能。

```
1  # ch11_39_4.py
2  def errcheck(func):                  # 装饰器
3      def newFunc(*args):
4          if args[1] != 0:
5              result = func(*args)
6          else:
7              result = "除数不可为0"
8          print('函数名称 : ', func.__name__)
9          print('函数参数 : ', args)
10         print('执行结果 : ', result)
11         return result
12     return newFunc
13 @errcheck                            # 设置装饰器
14 def mydiv(x, y):                     # 函数
15     return x/y
16
17 print(mydiv(6,2))
18 print(mydiv(6,0))
```

执行结果

```
==================== RESTART: D:\Python\ch11\ch11_39_4.py ====================
函数名称 :  mydiv
函数参数 :  (6, 2)
执行结果 :  3.0
3.0
函数名称 :  mydiv
函数参数 :  (6, 0)
执行结果 :  除数不可为0
除数不可为0
```

在上述程序第 3 行的 newFunc(*args) 中出现 *args，这会接收所传递的参数，同时以元组（tuple）方式存储，第 4 行是检查除数是否为 0，如果不为 0 则执行第 5 行除法运算，设置除法结果存在 result 变量中。如果第 4 行检查除数是 0 则执行第 7 行，设置 result 变量内容是“除数不可为 0”。

一个函数可以有两个以上的装饰器，方法是在函数上方设置装饰器函数，当有多个装饰器函数时，会按由下往上次序一次执行装饰器，这又称为**装饰器堆栈**（decorator stacking）。

程序实例 ch11_39_5.py：扩充设计 ch11_39_3.py 程序，主要是为 greeting( ) 函数增加 @bold 装饰器函数，这个函数会在字符串前后增加 bold 字符串。另外一个需注意的是，@bold 装饰器是在 @upper 装饰器的上方。

```
1  # ch11_39_5.py
2  def upper(func):                # 大写装饰器
3      def newFunc(args):
4          oldresult = func(args)
5          newresult = oldresult.upper()
6          return newresult
7      return newFunc
8  def bold(func):                 # 加粗体字符串装饰器
9      def wrapper(args):
10         return 'bold' + func(args) + 'bold'
11     return wrapper
12
13 @bold                           # 设置加粗体字符串装饰器
14 @upper                          # 设置大写装饰器
15 def greeting(string):           # 问候函数
16     return string
17
18 print(greeting('Hello! iPhone'))
```

执行结果

```
=================== RESTART: D:/Python/ch11/ch11_39_5.py ===================
boldHELLO! IPHONEbold
```

上述程序会先执行下方的 @upper 装饰器，这时可以将字符串改为大写，然后再执行 @bold 装饰器，最后得到前后增加 bold 的字符串。装饰器位置改变也将改变执行结果，可参考下列实例。

程序实例 ch11_39_6.py：更改 @upper 和 @bold 次序，重新设计 ch11_39_5.py，并观察执行结果。

```
1  # ch11_39_6.py
2  def upper(func):                # 装饰器
3      def newFunc(args):
4          oldresult = func(args)
5          newresult = oldresult.upper()
6          return newresult
7      return newFunc
8  def bold(func):
9      def wrapper(args):
10         return 'bold' + func(args) + 'bold'
11     return wrapper
12
13 @upper                          # 设置大写装饰器
14 @bold                           # 设置加粗体字符串大写装饰器
15 def greeting(string):           # 问候函数
16     return string
17
18 print(greeting('Hello! iPhone'))
```

执行结果

```
=================== RESTART: D:/Python/ch11/ch11_39_6.py ===================
BOLDHELLO! IPHONEBOLD
```

11-14 专题——函数的应用 / 最大公约数 / 质数

11-14-1 用函数重新设计记录一篇文章每个单词出现次数

程序实例 ch11_40.py：这个程序主要是设计两个函数，modifySong( ) 会将所传来的字符串有标点符号部分用空格符取代；wordCount( ) 会将字符串转成列表，同时将列表转成字典，最后遍历字典然后记录每个单词出现的次数。

```
1  # ch11_40.py
2  def modifySong(songStr):                # 将歌曲的标点符号用空字符取代
3      for ch in songStr:
4          if ch in ".,?":
5              songStr = songStr.replace(ch,'')
6      return songStr                      # 返回取代结果
7
8  def wordCount(songCount):
9      global mydict
10     songList = songCount.split()        # 将歌曲字符串转成列表
11     print("以下是歌曲列表")
12     print(songList)
13     mydict = {wd:songList.count(wd) for wd in set(songList)}
14
15 data = """Are you sleeping, are you sleeping, Brother John, Brother John?
16 Morning bells are ringing, morning bells are ringing.
17 Ding ding dong, Ding ding dong."""
18
19 mydict = {}                             # 空字典未来存储单词计数结果
20 print("以下是将歌曲大写字母全部改成小写同时将标点符号用空字符取代")
21 song = modifySong(data.lower())
22 print(song)
23
24 wordCount(song)                         # 执行歌曲单词计数
25 print("以下是最后执行结果")
26 print(mydict)                           # 打印字典
```

执行结果

```
===================== RESTART: D:\Python\ch11\ch11_40.py =====================
以下是将歌曲大写字母全部改成小写同时将标点符号用空字符取代
are you sleeping are you sleeping brother john brother john
morning bells are ringing morning bells are ringing
ding ding dong ding ding dong
以下是歌曲列表
['are', 'you', 'sleeping', 'are', 'you', 'sleeping', 'brother', 'john', 'brother
', 'john', 'morning', 'bells', 'are', 'ringing', 'morning', 'bells', 'are', 'rin
ging', 'ding', 'ding', 'dong', 'ding', 'ding', 'dong']
以下是最后执行结果
{'are': 4, 'you': 2, 'bells': 2, 'dong': 2, 'ringing': 2, 'brother': 2, 'john':
2, 'sleeping': 2, 'morning': 2, 'ding': 4}
```

11-14-2 最大公约数

在第 7 章习题 ex7_16.py 已经有介绍过**最大公约数**的概念了，有两个数字分别是 n1 和 n2，**公约数**是可以被 n1 和 n2 整除的数字，1 是它们的公约数，但不是最大公约数。假设最大公约数是 gcd，查找最大公约数可以从 n=2, 3, … 开始，每次找到比较大的公约数时将此 n 设给 gcd，直到 n 大于 n1 或 n2，最后的 gcd 值就是最大公约数。

程序实例 ch11_41.py：设计最大公约数 GCD 函数，然后输入两个数字做测试。

```
1  # ch11_41.py
2  def GCD(n1, n2):
3      gcd = 1                          # 初始化最大公约数
4      n = 2                            # 从2开始检测
5      while n <= n1 and n <= n2:
6          if n1 % n == 0 and n2 % n == 0:
7              gcd = n                  # 新最大公约数
8          n += 1
9      return gcd
10
11 n1, n2 = eval(input("请输入2个整数值 : "))
12 print("最大公约数是 : ", GCD(n1,n2))
```

执行结果

```
==================== RESTART: D:\Python\ch11\ch11_41.py ====================
请输入2个整数值 : 16, 24
最大公约数是 :  8
>>>
==================== RESTART: D:\Python\ch11\ch11_41.py ====================
请输入2个整数值 : 150, 2525
最大公约数是 :  25
```

11-14-3　质数

在 7-3-4 节有说明质数的概念与算法，本节将讲解设计质数的函数 isPrime( )。

程序实例 ch11_42.py：设计 isPrime( ) 函数，这个函数可以响应所输入的数字是否质数，如果是返回 True，否则返回 False。

```
1  # ch11_42.py
2  def isPrime(num):
3      """ 测试num是否质数 """
4      for n in range(2, num):
5          if num % n == 0:
6              return False
7      return True
8
9  num = int(input("请输入大于1的整数做质数测试 = "))
10 if isPrime(num):
11     print("%d是质数" % num)
12 else:
13     print("%d不是质数" % num)
```

执行结果

```
==================== RESTART: D:\Python\ch11\ch11_42.py ====================
请输入大于1的整数做质数测试 = 12
12不是质数
>>>
==================== RESTART: D:\Python\ch11\ch11_42.py ====================
请输入大于1的整数做质数测试 = 13
13是质数
```

习题

1. 请设计一个绝对值函数 absolute(n)，如果输入 -5 则输出 5，如果输入 5 则输出 5。（11-2 节）

```
===================== RESTART: D:\Python\ex\ex11_1.py =====================
请输入数值 = 6
绝对值是  6
>>>
===================== RESTART: D:\Python\ex\ex11_1.py =====================
请输入数值 = -11
绝对值是  11
```

2. 请设计 mymax(n1, n2)，此函数将输出较大值。(11-2 节)

```
===================== RESTART: D:\Python\ex\ex11_2.py =====================
请输入2个数值 = 10, 20
较大值是 :  20
>>>
===================== RESTART: D:\Python\ex\ex11_2.py =====================
请输入2个数值 = 9, 2
较大值是 :  9
```

3. 请设计一个函数 reverse(n)，此函数可以反向显示此数。(11-2 节)

```
===================== RESTART: D:\Python\ex\ex11_3.py =====================
请输入1个数值 = 5793
3975
```

4. 请设计可以执行两个数值运算的加法、减法、乘法、除法运算的小型计算器。这个程序必须设计 add(n1, n2)、sub(n1, n2)、mul(n1, n2)、div(n1, n2)4 个函数，所有计算结果必须使用 return 返回给主程序。(11-3 节)

```
===================== RESTART: D:\Python\ex\ex11_4.py =====================
请输入第1个数字 = 10
请输入第2个数字 = 5
请输入运算符(+,-,*,/) : +
计算结果 =  15
>>>
===================== RESTART: D:\Python\ex\ex11_4.py =====================
请输入第1个数字 = 10
请输入第2个数字 = 5
请输入运算符(+,-,*,/) : /
计算结果 =  2.0
>>>
===================== RESTART: D:\Python\ex\ex11_4.py =====================
请输入第1个数字 = 10
请输入第2个数字 = 5
请输入运算符(+,-,*,/) : @
运算公式输入错误
```

5. 请将上一题扩充为可以重复执行，每次运算结束会询问是否继续，如果输入 Y 或 y，程序继续，若是输入其他字符程序会结束。(11-3 节)

```
===================== RESTART: D:\Python\ex\ex11_5.py =====================
请输入第1个数字 = 10
请输入第2个数字 = 5
请输入运算符(+,-,*,/) : +
计算结果 =  15
是否继续?(Y or y=继续) : y
请输入第1个数字 = 10
请输入第2个数字 = 5
请输入运算符(+,-,*,/) : /
计算结果 =  2.0
是否继续?(Y or y=继续) : q
```

6. 请重新设计 ch11_14.py，请将 guest_info( ) 函数在传递参数不变的情况下，处理为适合外国人姓名的使用环境。这个程序使用以下两个数据做测试。(11-3 节)

```
firstname:Ivan        middlename:Carl        lastname:Hung
firstname:Mary        middlename:Ice         lastname:Hung
```

```
===================== RESTART: D:/Python/ex/ex11_6.py =====================
Mr. Ivan Carl Hung Welcome
Miss Mary Ice Hung Welcome
```

7. 请设计摄氏温度转华氏温度函数 CtoF(c)，华氏温度转摄氏温度函数 FtoC(f)，然后设计下列温度转换表。(11-3 节)

```
===================== RESTART: D:\Python\ex\ex11_7.py =====================
摄氏温度        华氏温度      |      华氏温度         摄氏温度
=================================================================
   21           69.80         |        70           21.11
   22           71.60         |        75           23.89
   23           73.40         |        80           26.67
   24           75.20         |        85           29.44
   25           77.00         |        90           32.22
   26           78.80         |        95           35.00
   27           80.60         |       100           37.78
   28           82.40         |       105           40.56
   29           84.20         |       110           43.33
   30           86.00         |       115           46.11
```

8. 在 7-6-3 节已经有介绍圆周率的莱布尼茨公式，如下所示：(11-3 节)

$$pi = 4\left(1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \cdots + \frac{(-1)^{i+1}}{2i-1}\right)$$

设计一个 pi(i) 函数，列出 i 是 1, 1001, … ,9001 时的 pi（i）值。

```
===================== RESTART: D:/Python/ex/ex11_8.py =====================
  i       PI
================
    1    4.00000
 1001    3.14259
 2001    3.14209
 3001    3.14193
 4001    3.14184
 5001    3.14179
 6001    3.14176
 7001    3.14174
 8001    3.14172
 9001    3.14170
```

9. 在第 4 章习题 ex4_12.py 中有说明计算三角形面积的方法，三角形边长的特点是两边长的和必须大于第三边。请设计 isTriangle(s1,s2,s3) 函数，这个函数可以判断所输入三角形的三个边长，可否成为三角形。如果所输入的边长可以成为三角形，同时设计 area(s1,s2,s3) 函数计算三角形的面积。(11-3 节)

```
===================== RESTART: D:\Python\ex\ex11_9.py =====================
请输入3个边长 : 5, 2, 2
这不是三角形的边长
>>>
===================== RESTART: D:\Python\ex\ex11_9.py =====================
请输入3个边长 : 2, 2, 2
这是三角形的边长
三角形面积是 :    1.732
```

10. 请设计一个函数 isPalindrome(n)，这个函数可以判断所输入的数值是不是回文（Palindrome）数字，回文数字的条件是从左读或是从右读都相同。例如，22,232,556655, … , 都算是回文数字。(11-3 节)

```
===================== RESTART: D:\Python\ex\ex11_10.py =====================
请输入1个数值 = 232
这是回文数
>>>
===================== RESTART: D:\Python\ex\ex11_10.py =====================
请输入1个数值 = 556655
这是回文数
>>>
===================== RESTART: D:\Python\ex\ex11_10.py =====================
请输入1个数值 = 5566
这不是回文数
```

11. 请重新设计 ch11_24.py，将程序改为制作 pizza，所以请将函数名称改为 make_pizza，第一

个参数改为 pizza 的尺寸，然后请到 pizza 店选择 5 种配料。(11-5 节)

```
===================== RESTART: D:\Python\ex\ex11_11.py =====================
这个  5  吋Pizza所加配料如下
---  海鲜
这个  7  吋Pizza所加配料如下
---  蔬菜
---  辛香料
---  香菇
---  干酪
---  海鲜
```

12. 设计一个递归函数 isPalindrome(s)，这个函数可以测试所输入的字符串是不是回文字符串，回文字符串的条件是从左读或是从右读都相同。例如，aa,aba,moom, … , 都算是回文字符串。(11-7 节)

```
===================== RESTART: D:\Python\ex\ex11_12.py =====================
请输入字符串 : aba
aba 是回文字符串
>>>
===================== RESTART: D:\Python\ex\ex11_12.py =====================
请输入字符串 : data
data 不是回文字符串
>>>
===================== RESTART: D:\Python\ex\ex11_12.py =====================
请输入字符串 : moom
moom 是回文字符串
```

13. Fibonacci 数列的起源最早可以追溯到 1150 年印度数学家 Gopala，在西方最早研究这个数列的是意大利科学家**列奥纳多 • 斐波那契**（Leonardo Fibonacci），后来人们将此数列简称为**费氏数列**。

请设计递归函数 fib(n)，产生前 10 个费氏数列 Fibonacci 数字，fib(n) 中的 n 主要是此数列的索引，费氏数列数字的规则如下。(11-7 节)

```
F0 = 0                          # 索引是 0
F1 = 1                          # 索引是 1
…
Fn = Fn-1 + Fn-2 ( n ≥ 2)       # 索引是 n
```

$F_0 = 0$　　# 索引是 0

$F_1 = 1$　　# 索引是 1

…

$F_n = F_{n-1} + F_{n-2}\ (n \geq 2)$　　# 索引是 n

最后值应该是 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, …

```
===================== RESTART: D:\Python\ex\ex11_13.py =====================
下列是前10个Fibonacci数列
  0   1   1   2   3   5   8  13  21  34
```

14. 重新设计 ch11_34.py，产生偶数列表。(11-9 节)

```
===================== RESTART: D:\Python\ex\ex11_14.py =====================
偶数列表:  [10, 20, 30]
```

15. 重新设计 ch11_36.py，产生偶数列表。(11-9 节)

```
===================== RESTART: D:\Python\ex\ex11_15.py =====================
偶数列表:  [10, 20, 30]
```

16. 美国 NBA 球员 Lin 的前 10 场得分资料如下 :(11-9 节)

25, 18, 12, 22, 31, 17, 26, 19, 18, 10

请使用匿名函数和 filter( ) 函数，列出得分超过 20 分（含）的列表。

```
===================== RESTART: D:\Python\ex\ex11_16.py =====================
得分大于或等于20分的列表:  [25, 22, 31, 26]
```

17. 请重新设计 ch11_39_5.py，增加设计 @italic 装饰器，这个装饰器可以在字符串外围增加 italic 字符串，下列是执行结果。(11-9 节)

```
====================== RESTART: D:/Python/ex/ex11_17.py ======================
italicboldHELLO! IPHONEbolditalic
```

18. 使用 map( ) 将 [1,2,3,4,5] 转为 ['1','2','3','4','5']。

```
====================== RESTART: D:/Python/ex/ex11_18.py ======================
['1', '2', '3', '4', '5']
```

12

第 1 2 章

类——面向对象的程序设计

本章摘要

12-1 类的定义与使用
12-2 类的访问权限——封装
12-3 类的继承
12-4 多态
12-5 多重继承
12-6 type 与 instance
12-7 特殊属性
12-8 类的特殊方法
12-9 专题——几何数据的应用

Python 其实是一种面向对象（Object Oriented Programming）语言，在 Python 中所有的数据类型都是对象，Python 也允许程序设计师自创数据类型，这种自创的数据类型就是本章的主题——类（class）。

设计程序时可以将世间万物分组归类，然后使用类（class）来定义分类，本章将列举一系列不同的类，扩展读者的思维。

12-1 类的定义与使用

类的语法定义如下：

```
class     Classname( )          # 类名称第一个字母建议使用大写
    statement1
    …
    statementn
```

本节将以银行为例，说明最基本的类的概念。

12-1-1 定义类

程序实例 ch12_1.py：Banks 类的定义。

```
1  # ch12_1.py
2  class Banks():
3      ''' 定义银行类 '''
4      bankname = 'Taipei Bank'          # 定义属性
5      def motto(self):                  # 定义方法
6          return "以客为尊"
```

执行结果 这个程序没有输出结果。

对上述程序而言，Banks 是**类名称**，在这个类中定义了一个**属性** bankname 与一个**方法** motto。

在类内定义方法（method）的方式与第 11 章定义函数的方式相同，但是不可以称之为函数（function）而必须称之为**方法**（method），在程序设计时可以随时调用函数，但是只有属于该类的**对象**（object）才可调用相关的方法。

12-1-2 操作类的属性与方法

若是想操作类的属性与方法，首先需声明该类的**对象**（object）**变量**，可以简称**对象**，然后使用下列方式操作。

```
object.类的属性
object.类的方法( )
```

程序实例 ch12_2.py：扩充 ch12_1.py，列出银行的名称与服务宗旨。

```
1  # ch12_2.py
2  class Banks():
3      ''' 定义银行类 '''
4      bankname = 'Taipei Bank'     # 定义属性
5      def motto(self):             # 定义方法
6          return "以客为尊"
7
8  userbank = Banks()               # 定义对象userbank
9  print("目前服务银行是 ", userbank.bankname)
10 print("银行服务理念是 ", userbank.motto())
```

执行结果

```
===================== RESTART: D:\Python\ch12\ch12_2.py =====================
目前服务银行是  Taipei Bank
银行服务理念是  以客为尊
```

从上述执行结果可以发现，我们成功地存取了 Banks 类内的**属性**与**方法**。程序第 8 行定义了 userbank 当作 Banks 类的对象，然后使用 userbank 对象读取了 Banks 类内的 bankname 属性与 motto( ) 方法。这个程序主要是列出 bankname 属性值与 motto( ) 方法返回的内容。

建立一个对象后，这个对象就可以像其他 Python 对象一样，可以将这个对象当作列表、元组、字典或集合元素使用，也可以将此对象当作函数的参数传送，或是将此对象当作函数的返回值。

12-1-3　类的建构方法

建立类很重要的一个工作是**初始化整个类**。**初始化**类是在类内建立一个初始化**方法**（method），这是一个特殊**方法**，当在程序内声明这个类的对象时将自动执行这个方法。初始化方法有一个固定名称是“__init__( )”，写法是 init 左右各有两个下画线字符。init 其实是 initialization 的缩写，通常又将这类初始化的方法称为**建构方法**（constructor）。在初始化的方法内可以执行一些属性变量设置。下面先用一个实例做解说。

程序实例 ch12_3.py：重新设计 ch12_2.py，设置初始化方法，同时存第一笔开户的钱 100 元到银行里，然后列出存款金额。

```
1  # ch12_3.py
2  class Banks():
3      ''' 定义银行类 '''
4      bankname = 'Taipei Bank'                # 定义属性
5      def __init__(self, uname, money):       # 初始化方法
6          self.name = uname                   # 设置存款者名字
7          self.balance = money                # 设置所存的钱
8
9      def get_balance(self):                  # 获得存款余额
10         return self.balance
11
12 hungbank = Banks('hung', 100)               # 定义对象hungbank
13 print(hungbank.name.title(), " 存款余额是 ", hungbank.get_balance())
```

执行结果

```
===================== RESTART: D:\Python\ch12\ch12_3.py =====================
Hung  存款余额是  100
```

程序 12 行定义 Banks 类的 hungbank 对象时，Banks 类会自动启动 __init__( ) 初始化函数，在这个定义中 self 是必需的，同时需放在所有参数的**最前面**（相当于最左边），Python 在初始化时会自动传入这个参数 self，代表的是类本身的对象，未来在类内想要参照各**属性**与**函数**执行运算都要使用 self，可参考第 6、7 和 10 行。

在这个 Banks 类的 __init__(self, uname, money) 方法中，有另外两个参数 uname 和 money，未来在定义 Banks 类的对象时（第 12 行）需要传递两个参数，分别给 uname 和 money。至于程序第 6 和 7 行内容如下：

```
self.name = uname              ; name 是 Banks 类的属性
self.balance = money           ; balance 是 Banks 类的属性
```

读者可能会思考，既然 __init__ 这么重要，为何 ch12_2.py 没有这个初始化函数仍可运行？其实对 ch12_2.py 而言是使用预设没有参数的 __init__( ) 方法。

在程序第 9 行另外有一个 get_balance(self) 方法，在这个方法内只有一个参数 self，所以调用时可以不用任何参数，可以参考第 13 行。这个方法目的是返回存款余额。

程序实例 ch12_4.py：扩充 ch12_3.py，主要是增加执行存款与取款功能，同时在类内可以直接列出目前余额。

```
1  # ch12_4.py
2  class Banks():
3      ''' 定义银行类 '''
4      bankname = 'Taipei Bank'                   # 定义属性
5      def __init__(self, uname, money):          # 初始化方法
6          self.name = uname                      # 设置存款者名字
7          self.balance = money                   # 设置所存的钱
8
9      def save_money(self, money):               # 设计存款方法
10         self.balance += money                  # 执行存款
11         print("存款 ", money, " 完成")          # 打印存款完成
12
13     def withdraw_money(self, money):           # 设计取款方法
14         self.balance -= money                  # 执行取款
15         print("取款 ", money, " 完成")          # 打印取款完成
16
17     def get_balance(self):                     # 获得存款余额
18         print(self.name.title(), " 目前余额: ", self.balance)
19
20 hungbank = Banks('hung', 100)                  # 定义对象hungbank
21 hungbank.get_balance()                         # 获得存款余额
22 hungbank.save_money(300)                       # 存款300元
23 hungbank.get_balance()                         # 获得存款余额
24 hungbank.withdraw_money(200)                   # 提款200元
25 hungbank.get_balance()                         # 获得存款余额
```

执行结果

```
==================== RESTART: D:\Python\ch12\ch12_4.py ====================
Hung  目前余额:  100
存款  300  完成
Hung  目前余额:  400
取款  200  完成
Hung  目前余额:  200
```

类建立完成后，随时可以使用多个对象引用这个类的属性与函数，可参考下列实例。

程序实例 ch12_5.py：使用与 ch12_4.py 相同的 Banks 类，然后定义两个对象操作这个类。下面是与 ch12_4.py 不同的程序代码内容。

```
20  hungbank = Banks('hung', 100)              # 定义对象hungbank
21  johnbank = Banks('john', 300)              # 定义对象johnbank
22  hungbank.get_balance()                     # 获得hung存款余额
23  johnbank.get_balance()                     # 获得john存款余额
24  hungbank.save_money(100)                   # hung存款100
25  johnbank.withdraw_money(150)               # john取款150
26  hungbank.get_balance()                     # 获得hung存款余额
27  johnbank.get_balance()                     # 获得john存款余额
```

执行结果

```
===================== RESTART: D:\Python\ch12\ch12_5.py =====================
Hung  目前余额:  100
John  目前余额:  300
存款  100  完成
取款  150  完成
Hung  目前余额:  200
John  目前余额:  150
```

12-1-4　属性初始值的设置

在先前程序的 Banks 类中第 4 行 bankname 是设为“Taipei Bank”，其实这是初始值的设置，通常 Python 在设初始值时是将初始值设在 __init__( ) 方法内，下列这个程序在定义 Banks 类对象时，省略开户金额，相当于定义 Banks 类对象时只要两个参数。

程序实例 ch12_6.py：设置开户（定义 Banks 类对象）只要姓名，同时设置开户金额是 0 元，读者可留意第 7 和 8 行的设置。

```
1   # ch12_6.py
2   class Banks():
3       ''' 定义银行类 '''
4   
5       def __init__(self, uname):               # 初始化方法
6           self.name = uname                    # 设置存款者名字
7           self.balance = 0                     # 设置开户金额是0
8           self.bankname = "Taipei Bank"        # 设置银行名称
9   
10      def save_money(self, money):             # 设计存款方法
11          self.balance += money                # 执行存款
12          print("存款 ", money, " 完成")        # 打印存款完成
13  
14      def withdraw_money(self, money):         # 设计取款方法
15          self.balance -= money                # 执行取款
16          print("取款 ", money, " 完成")        # 打印取款完成
17  
18      def get_balance(self):                   # 获得存款余额
19          print(self.name.title(), " 目前余额: ", self.balance)
20  
21  hungbank = Banks('hung')                     # 定义对象hungbank
22  print("目前开户银行 ", hungbank.bankname)     # 列出目前开户银行
23  hungbank.get_balance()                       # 获得hung存款余额
24  hungbank.save_money(100)                     # hung存款100
25  hungbank.get_balance()                       # 获得hung存款余额
```

执行结果

```
===================== RESTART: D:\Python\ch12\ch12_6.py =====================
目前开户银行  Taipei Bank
Hung  目前余额:  0
存款  100  完成
Hung  目前余额:  100
```

12-2 类的访问权限——封装

可以看到我们可以从程序直接引用类内的属性（可参考 ch12_6.py 的第 22 行）与方法（可参考 ch12_6.py 的第 23 行），像这种类内的属性可以让外部引用的称为**公有（public）属性**，而可以让外部引用的方法称为**公有方法**。前面所使用的 Banks 类内的属性与方法都是**公有属性与方法**。但是设计程序时可以发现，外部直接引用时也代表可以直接修改类内的属性值，这将造成类数据不安全。

Python 提供了**私有属性**与**方法**的概念，这个概念的主要思想是类外无法直接更改类内的**私有属性**，类外也无法直接调用**私有方法**，这个概念又称为**封装**（encapsulation）。

12-2-1 私有属性

为了确保类内属性的安全，其实有必要限制外部无法直接存取类内的属性值。

程序实例 ch12_7.py：外部直接存取属性值，造成存款余额不安全的实例。

```
21  hungbank = Banks('hung')                    # 定义对象hungbank
22  hungbank.get_balance()
23  hungbank.balance = 10000                    # 类外直接篡改存款余额
24  hungbank.get_balance()
```

执行结果

```
===================== RESTART: D:\Python\ch12\ch12_7.py =====================
Hung  目前余额:  0
Hung  目前余额:  10000
```

上述程序第 23 行直接在类外就更改了存款余额，当第 24 行列出存款余额时，可以发现没有经过 Banks 类内的 save_money( ) 方法存钱动作，整个余额就从 0 元增至 10000 元。为了避免这种现象产生，Python 对于类内的属性增加了**私有属性**（private attribute）的概念，应用方式是声明时在属性名称前面增加 __ （**两个下画线**）。声明为**私有属性**后，类外的程序就无法引用了。

程序实例 ch12_8.py：重新设计 ch12_7.py，主要是将 Banks 类的属性声明为私有属性，这样就无法由外部程序修改了。

```
1   # ch12_8.py
2   class Banks():
3       ''' 定义银行类 '''
4   
5       def __init__(self, uname):              # 初始化方法
6           self.__name = uname                 # 设置私有存款者名字
7           self.__balance = 0                  # 设置私有开户金额是0
8           self.__bankname = "Taipei Bank"     # 设置私有银行名称
9   
10      def save_money(self, money):            # 设计存款方法
11          self.__balance += money             # 执行存款
12          print("存款 ", money, " 完成")      # 打印存款完成
13  
14      def withdraw_money(self, money):        # 设计取款方法
15          self.__balance -= money             # 执行取款
16          print("取款 ", money, " 完成")      # 打印取款完成
17  
18      def get_balance(self):                  # 获得存款余额
19          print(self.__name.title(), " 目前余额: ", self.__balance)
20  
21  hungbank = Banks('hung')                    # 定义对象hungbank
22  hungbank.get_balance()
23  hungbank.__balance = 10000                  # 类外直接修改存款余额
24  hungbank.get_balance()
```

执行结果

```
===================== RESTART: D:\Python\ch12\ch12_8.py =====================
Hung  目前余额:  0
Hung  目前余额:  0
```

请读者留意第 6 ～ 8 行设置私有属性的方式。第 23 行尝试修改存款余额，但从输出结果可以知道修改失败，因为执行结果的存款余额是 0。对上述程序而言，存款余额只会在存款（save_money( )）和取款（withdraw_money( )）方法被触发时，依参数金额更改。

下面是执行完 ch12_8.py 后，尝试设置私有属性结果失败的实例。

```
>>> hungbank._Banks_balance = 12000
>>> hungbank.get_balance()
Hung  目前余额:  0
```

其实 Python 的高手可以用其他方式设置或取得私有属性，若是以执行完 ch12_8.py 之后为例，可以使用下列方法存取私有属性。

对象名称 ._ 类名称**私有属性**　**# 此例相当于 hungbank._Banks__balance**

下面是执行结果。

```
>>> hungbank.Banks__balance = 12000
>>> hungbank.get_balance()
Hung  目前余额:  0
```

实质上私有属性因为可以被外界调用，所以设置私有属性名称时就需特别小心。

12-2-2　私有方法

既然类有**私有属性**，其实也有**私有方法**（private method），它的概念与私有属性类似，基本思想是类外的程序无法调用。不过请留意实质上类外依旧可以调用此私有方法。至于其声明定义方式与私有属性相同，只要在方法前面加上 __（两个下画线）符号即可。若是延续上述程序实例，可能会遇上换汇的问题，通常银行在换汇时会针对客户对银行的贡献制定不同的汇率与手续费，这个部分是客户无法得知的，碰上这类应用就很适合以私有方法处理换汇程序。为了简化问题，下面是在初始化类时，先设置美金与台币的汇率以及换汇的手续费，其中，汇率（__rate）与手续费率（__service_charge）都是私有属性。

```
9           self.__rate = 30                        # 默认美金与台币换汇比例
10          self.__service_charge = 0.01            # 换汇的服务费
```

下面是使用者可以调用的公有方法，在这里只能输入换汇率的金额。

```
23      def usa_to_taiwan(self, usa_d):             # 美金兑换台币方法
24          self.result = self.__cal_rate(usa_d)
25          return self.result
```

在上述公有方法中调用了 __cal_rate（usa_d），这是**私有方法**，类外无法使用，下面是此**私有方法**的内容。

```
27      def __cal_rate(self,usa_d):                 # 计算换汇，这是私有方法
28          return int(usa_d * self.__rate * (1 - self.__service_charge))
```

在上述私有方法中可以看到内部包含比较敏感且不适合给外部人参与的数据。

程序实例 ch12_9.py：下面是私有方法应用的完整程序代码实例。

```
1  # ch12_9.py
2  class Banks():
3      ''' 定义银行类 '''
4  
5      def __init__(self, uname):              # 初始化方法
6          self.__name = uname                 # 设置私有存款者名字
7          self.__balance = 0                  # 设置私有开户金额是0
8          self.__bankname = "Taipei Bank"     # 设置私有银行名称
9          self.__rate = 30                    # 预设美金与台币换汇比例
10         self.__service_charge = 0.01        # 换汇的服务费
11 
12     def save_money(self, money):            # 设计存款方法
13         self.__balance += money             # 执行存款
14         print("存款 ", money, " 完成")       # 打印存款完成
15 
16     def withdraw_money(self, money):        # 设计取款方法
17         self.__balance -= money             # 执行取款
18         print("取款 ", money, " 完成")       # 打印取款完成
19 
20     def get_balance(self):                  # 获得存款余额
21         print(self.__name.title(), " 目前余额: ", self.__balance)
22 
23     def usa_to_taiwan(self, usa_d):         # 美金兑换台币方法
24         self.result = self.__cal_rate(usa_d)
25         return self.result
26 
27     def __cal_rate(self,usa_d):             # 计算换汇这是私有方法
28         return int(usa_d * self.__rate * (1 - self.__service_charge))
29 
30 hungbank = Banks('hung')                    # 定义对象hungbank
31 usdallor = 50
32 print(usdallor, " 美金可以兑换 ", hungbank.usa_to_taiwan(usdallor), " 台币")
```

执行结果

```
===================== RESTART: D:\Python\ch12\ch12_9.py =====================
50  美金可以兑换  1485  台币
```

如果类外直接调用私有属性会产生错误，当执行完 ch12_9.py 后，请执行下列指令。

```
>>> hungbank.__cal_rate(50)
Traceback (most recent call last):
  File "<pyshell#9>", line 1, in <module>
    hungbank.__cal_rate(50)
AttributeError: 'Banks' object has no attribute '__cal_rate'
```

破解私有方法的方式类似破解私有属性，当执行完 ch12_9.py 后，可以执行下列指令，直接计算汇率。

```
>>> hungbank._Banks__cal_rate(50)
1485
```

12-2-3　从存取属性值看 Python 风格 property( )

经过前两节的说明，相信读者对于 Python 的面向对象程序封装设计有了一些基础了，本节将讲解偏向 Python 风格的操作。为了容易说明与了解，本节将用简单的实例解说。

程序实例 ch12_9_1.py：定义成绩类 Score，这时外部可以打印与修改成绩。

```
1  # ch12_9_1.py
2  class Score():
3      def __init__(self, score):
4          self.score = score
5
6  stu = Score(50)
7  print(stu.score)
8  stu.score = 100
9  print(stu.score)
```

执行结果

```
==================== RESTART: D:/Python/ch12/ch12_9_1.py ====================
50
100
```

由于外部可以随意更改成绩，所以这是有风险、不恰当的。为了保护成绩，可以将分数设为私有属性，同时未来改成 getter 和 setter 存取这个私有属性。

程序实例 ch12_9_2.py：将 score 设为私有属性，设计含 getter 概念的 getscore( ) 和 setter 概念的 setscore( ) 存取分数，这时外部将无法直取存取 score。

```
1  # ch12_9_2.py
2  class Score():
3      def __init__(self, score):
4          self.__score = score
5      def getscore(self):
6          print("inside the getscore")
7          return self.__score
8      def setscore(self, score):
9          print("inside the setscore")
10         self.__score = score
11
12 stu = Score(0)
13 print(stu.getscore())
14 stu.setscore(80)
15 print(stu.getscore())
```

执行结果

```
==================== RESTART: D:/Python/ch12/ch12_9_2.py ====================
inside the getscore
0
inside the setscore
inside the getscore
80
```

如果外部强制修订私有属性 score，将不会成功。下面想在外部更改 score 为 100，但是失败了。

```
>>> stu.score = 100
>>> stu.getscore()
inside the getscore
80
```

上述语句虽然可以运行，但是新式 Python 设计风格是使用 property( ) 方法：

新式属性 = `property(getter[,setter[,fdel[,doc]]])`

getter 是获取属性值函数，setter 是设置属性值函数，fdel 是删除属性值函数，doc 是属性描述，返回的是新式属性，未来可以由此新式属性存取私有属性内容。

程序实例 ch12_19_3.py：使用 Python 风格重新设计 ch12_19_2.py，读者需留意第 11 行的 property( )，在这里设置 sc 当作 property( ) 的返回值，未来可以直接由 sc 存取私有属性 __score。

```
1  # ch12_9_3.py
2  class Score():
3      def __init__(self, score):
4          self.__score = score
5      def getscore(self):
6          print("inside the getscore")
7          return self.__score
8      def setscore(self, score):
9          print("inside the setscore")
10         self.__score = score
11     sc = property(getscore, setscore)   # Python 风格
12
13 stu = Score(0)
14 print(stu.sc)
15 stu.sc = 80
16 print(stu.sc)
```

执行结果

```
==================== RESTART: D:/Python/ch12/ch12_9_3.py ====================
inside the getscore
0
inside the setscore
inside the getscore
80
```

上述执行第 14 行时相当于执行 getscore( )，执行第 15 行时相当于执行 setscore( )。此外，虽然改用 property( ) 让工作呈现 Python 风格，但是在主程序中仍可以使用 getscore( ) 和 setscore( ) 方法。

12-2-4 装饰器 @property

延续前一节的讨论，我们可以使用装饰器 @property，首先将 getscore( ) 和 setscore( ) 方法的名称全部改为 sc( )，然后在 sc( ) 方法前加上下列装饰器。

（1）@property：放在 getter 方法前。

（2）@sc.setter：放在 setter 方法前。

程序实例 ch12_9_4.py：使用装饰器重新设计 ch12_9_3.py。

```
1  # ch12_9_4.py
2  class Score():
3      def __init__(self, score):
4          self.__score = score
5      @property
6      def sc(self):
7          print("inside the getscore")
8          return self.__score
9      @sc.setter
10     def sc(self, score):
11         print("inside the setscore")
12         self.__score = score
13
14 stu = Score(0)
15 print(stu.sc)
16 stu.sc = 80
17 print(stu.sc)
```

执行结果 与 ch12_9_3.py 相同。

经上述设计后未来将无法存取私有属性。

```
>>> stu.__score
Traceback (most recent call last):
  File "<pyshell#71>", line 1, in <module>
    stu.__score
AttributeError: 'Score' object has no attribute '__score'
```

上述语句只是将 sc 特性应用在 Score 类内的属性 __score，其实这个概念可以扩充至一般程序设计中，例如，计算面积。

程序实例 ch12_9_5.py：计算正方形的面积。

```
1  # ch12_9_5.py
2  class Square():
3      def __init__(self, sideLen):
4          self.__sideLen = sideLen
5      @property
6      def area(self):
7          return self.__sideLen ** 2
8
9  obj = Square(10)
10 print(obj.area)
```

执行结果

```
==================== RESTART: D:/Python/ch12/ch12_9_5.py ====================
100
```

12-2-5　方法与属性的类型

严格区分设计 Python 面向对象程序时，又可将类的方法区分为实例方法（属性）与类方法（属性）。

实例方法与属性的特色是有 self，属性开头是 self，同时所有方法的第一个参数是 self，这些是建立类对象时，属于对象的一部分。先前所述的都是实例方法与属性，使用时需建立此类的对象，然后由对象调用。

类方法前面则是 @classmethod，所不同的是第一个参数习惯是用 cls。类方法与属性不需要实例化，它们可以由类本身直接调用。另外，类属性会随时被更新。

程序实例 ch12_9_6.py：类方法与属性的应用。这个程序执行时，每次建立 Counter( ) 类对象（11 ～ 13 行），类属性值会更新，此外，这个程序使用类名称就可以直接调用类属性与方法。

```
1  # ch12_9_6.py
2  class Counter():
3      counter = 0                              # 类属性,可由类本身调用
4      def __init__(self):
5          Counter.counter += 1                 # 更新指标
6      @classmethod
7      def show_counter(cls):                   # 类方法,可由类本身调用
8          print("class method")
9          print("counter = ", cls.counter)     # 也可使用Counter.counter调用
10         print("counter = ", Counter.counter)
11
12 one = Counter()
13 two = Counter()
14 three = Counter()
15 Counter.show_counter()
```

执行结果

```
==================== RESTART: D:/Python/ch12/ch12_9_6.py ====================
class method
counter =  3
counter =  3
```

12-2-6　静态方法

静态方法是由 @staticmethod 开头，不需要原先的 self 或 cls 参数，只是碰巧存在类的函数，与类方法和实例方法没有绑定关系，这个方法也是由类名称直接调用的。

程序实例 ch12_9_7.py：静态方法的调用实例。

```
1  # ch12_9_7.py
2  class Pizza():
3      @staticmethod
4      def demo():
5          print("I like Pizza")
6
7  Pizza.demo()
```

执行结果

```
==================== RESTART: D:/Python/ch12/ch12_9_7.py ====================
I like Pizza
```

12-3　类的继承

在程序设计时有时我们感觉某些类已经大致可以满足需求，这时可以修改此类完成工作，可是这样会让程序显得更复杂。或者可以重新写新的类，可是这样会需要维护更多的程序。

碰上这类问题解决的方法是使用继承，也就是延续使用旧类，设计子类继承此类，然后在子类中设计新的属性与方法，这也是本节的主题。

在面向对象程序设计中，类是可以继承的，其中，被继承的类称为**父类**（parent class）、**基类**（base class）或**超类**（superclass），继承的类称为**子类**（child class）或**衍生类**（derived class）。类继承的最大优点是许多**父类的公有方法**或**属性**，在子类中不用重新设计，可以直接引用。



在设计程序时，基类必须在衍生类前面，整个程序代码结构如下。

```
class BaseClassName():                          # 先定义基类
    Base Class 的内容
class DerivedClassName(BaseClassName):          # 再定义衍生类
    Derived Class 的内容
```

衍生类继承了基类的公有属性与方法，同时也可以有自己的属性与方法。

12-3-1　衍生类继承基类的实例应用

在延续先前说明的 Banks 类前，下面先用简单的范例做说明。

程序实例 ch12_9_8.py：设计 Father 类，也设计 Son 类，Son 类继承了 Father 类，Father 类有 hometown( ) 方法，然后 Father 类和 Son 类对象都会调用 hometown( ) 方法。

```
1  # ch12_9_8.py
2  class Father():
3      def hometown(self):
4          print('我住在台北')
5
6  class Son(Father):
7      pass
8
9  hung = Father()
10 ivan = Son()
11 hung.hometown()
12 ivan.hometown()
```

执行结果

```
==================== RESTART: D:\Python\ch12\ch12_9_8.py ====================
我住在台北
我住在台北
```

上述 Son 类继承了 Father 类，所以第 12 行可以调用 Father 类，然后可以打印相同的字符串。

程序实例 ch12_10.py：延续 Banks 类建立一个分行 Shilin_Banks，这个衍生类没有任何数据，直接引用基类的公有函数，执行银行的存款作业。下面是与 ch12_9.py 不同的程序代码。

```
30 class Shilin_Banks(Banks):
31     # 定义士林分行
32     pass
33
34 hungbank = Shilin_Banks('hung')              # 定义对象hungbank
35 hungbank.save_money(500)
36 hungbank.get_balance()
```

执行结果

```
===================== RESTART: D:\Python\ch12\ch12_10.py =====================
存款  500  完成
Hung  目前余额:  500
```

上述第 35 和 36 行所引用的方法就是基类 Banks 的公有方法。

12-3-2　如何取得基类的私有属性

基于保护的原因，基本上类定义外是无法直接取得类内的**私有属性**的，即使是它的衍生类也无法直接读取，如果真是要取得可以使用 return 方式，返回私有属性内容。

在延续先前的 Banks 类前，下面先用短小易懂的程序讲解这个概念。

程序实例 ch12_10_1.py：设计一个子类 Son 的对象存取父类私有属性的应用。

```
1  # ch12_10_1.py
2  class Father():
3      def __init__(self):
4          self.__address = '台北市罗斯福路';
5      def getaddr(self):
6          return self.__address
7
8  class Son(Father):
9      pass
10
11 hung = Father()
12 ivan = Son()
13 print('父类 : ',hung.getaddr())
14 print('子类 : ',ivan.getaddr())
```

执行结果

```
==================== RESTART: D:\Python\ch12\ch12_10_1.py ====================
父类 :  台北市罗斯福路
子类 :  台北市罗斯福路
```

从上述第 14 行可以看到，子类对象 ivan 顺利地取得父类的私有属性 address。

程序实例 ch12_11.py：衍生类对象取得基类的银行名称 bankname 属性。

```
30      def bank_title(self):                # 获得银行名称
31          return self.__bankname
32
33  class Shilin_Banks(Banks):
34      # 定义士林分行
35      pass
36
37  hungbank = Shilin_Banks('hung')          # 定义对象hungbank
38  print("我的存款银行是: ", hungbank.bank_title())
```

执行结果

```
===================== RESTART: D:\Python\ch12\ch12_11.py =====================
我的存款银行是:  Taipei Bank
```

12-3-3　衍生类与基类有相同名称的属性

程序设计时，衍生类也可以有自己的初始化 __init__( ) 方法，同时也有可能衍生类的属性与方法名称和基类重复，碰上这个状况 Python 会先找寻衍生类是否有这个名称，如果有则先使用，如果没有则使用基类的名称内容。

程序实例 ch12_11_1.py：衍生类与基类有相同名称的简单说明。

```
1  # ch12_11_1.py
2  class Person():
3      def __init__(self,name):
4          self.name = name
5  class LawerPerson(Person):
6      def __init__(self,name):
7          self.name = name + "律师"
8
9  hung = Person("洪锦魁")
10 lawer = LawerPerson("洪锦魁")
11 print(hung.name)
12 print(lawer.name)
```

执行结果

```
==================== RESTART: D:\Python\ch12\ch12_11_1.py ====================
洪锦魁
洪锦魁律师
```

上述衍生类与基类有相同的属性 name，但是衍生类对象将使用自己的属性。下列是 Banks 类的应用说明。

程序实例 ch12_12.py：这个程序主要是将 Banks 类的 bankname 属性改为公有属性，但是在衍生类中则有自己的初始化方法，主要是基类与衍生类均有 bankname 属性，不同类对象将呈现不同的结果。下面是第 8 行的内容。

```
8          self.bankname = "Taipei Bank"       # 设置银行名称
```

下面是修改部分程序代码内容。

```
33 class Shilin_Banks(Banks):
34     # 定义士林分行
35     def __init__(self, uname):
36         self.bankname = "Taipei Bank - Shilin Branch"  # 定义分行名称
37
38 jamesbank = Banks('James')                             # 定义Banks类对象
39 print("James's banks = ", jamesbank.bankname)          # 打印银行名称
40 hungbank = Shilin_Banks('Hung')                        # 定义Shilin_Banks类对象
41 print("Hung's banks  = ", hungbank.bankname)           # 打印银行名称
```

执行结果

```
===================== RESTART: D:\Python\ch12\ch12_12.py =====================
James's banks =  Taipei Bank
Hung's banks  =  Taipei Bank - Shilin Branch
```

从上述可知，Banks 类对象 James 所使用的 bankname 属性是 Taipei Bank，Shilin_Banks 对象 Hung 所使用的 bankname 属性是 Taipei Bank – Shilin Branch。

12-3-4　衍生类与基类有相同名称的方法

程序设计时，衍生类也可以有自己的方法，同时也有可能衍生类的方法名称和基类方法名称重复，碰上这个状况 Python 会先寻找衍生类是否有这个名称，如果有则先使用，如果没有则使用基类的名称内容。

程序实例 ch12_12_1.py：衍生类的方法名称和基类方法名称重复的应用。

```
1  # ch12_12_1.py
2  class Person():
3      def job(self):
4          print("我是老师")
5
6  class LawerPerson(Person):
7      def job(self):
8          print("我是律师")
9
10 hung = Person()
11 ivan = LawerPerson()
12 hung.job()
13 ivan.job()
```

执行结果

```
==================== RESTART: D:\Python\ch12\ch12_12_1.py ====================
我是老师
我是律师
```

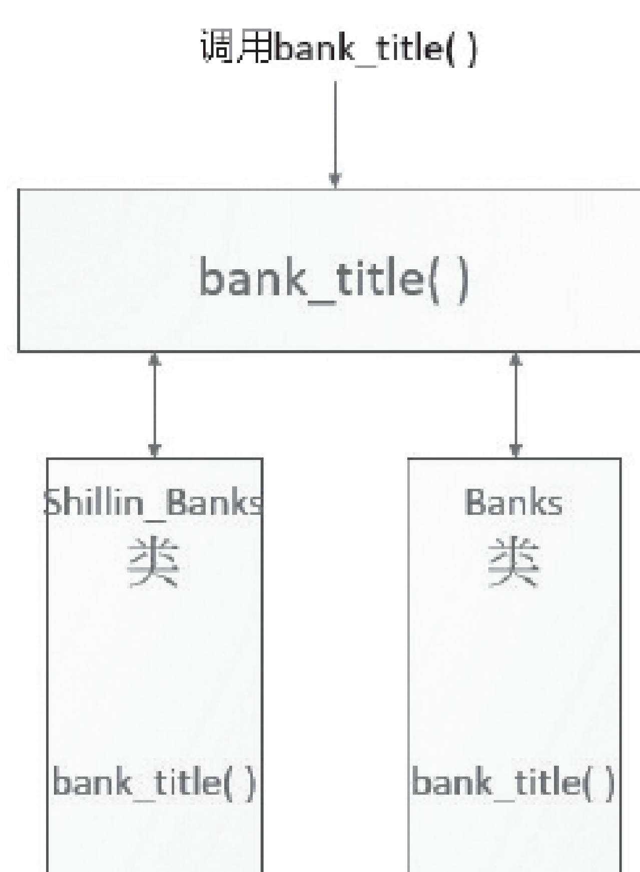
程序实例 ch12_13.py：衍生类与基类名称重复的实例。这个程序的基类与衍生类均有 bank_title( ) 函数，Python 会由触发 bank_title( ) 方法的对象去判别应使用哪一个方法执行。

```
30      def bank_title(self):                        # 获得银行名称
31          return self.__bankname
32
33  class Shilin_Banks(Banks):
34      # 定义士林分行
35      def __init__(self, uname):
36          self.bankname = "Taipei Bank - Shilin Branch"  # 定义分行名称
37      def bank_title(self):                        # 获得银行名称
38          return self.bankname
39
40  jamesbank = Banks('James')                       # 定义Banks类对象
41  print("James's banks = ", jamesbank.bank_title())  # 打印银行名称
42  hungbank = Shilin_Banks('Hung')                  # 定义Shilin_Banks类对象
43  print("Hung's banks  = ", hungbank.bank_title())   # 打印银行名称
```

执行结果

```
==================== RESTART: D:\Python\ch12\ch12_13.py ====================
James's banks =  Taipei Bank
Hung's banks  =  Taipei Bank - Shilin Branch
```

上述程序的执行过程如下。



上述第 30 行的 bank_title( ) 是属于 Banks 类，第 37 行的 bank_title( ) 是属于 Shilin_Banks 类。第 40 行是 Banks 对象，所以第 41 行会触发第 30 行的 bank_title( ) 方法。第 42 行是 Shilin_

Banks 对象，所以第 42 行会触发第 37 行的 bank_title( ) 方法。其实上述方法就是面向对象的**多态**（polymorphism），但是**多态**不一定需要是有父子关系的类。读者可以将以上想成**方法多功能化**，即相同的函数名称，放入不同类型的对象可以产生不同的结果。使用者可以不需要知道是如何设计的，隐藏在内部的设计细节交由程序设计师负责。12-4 节还会举实例说明。

12-3-5　衍生类引用基类的方法

衍生类引用基类的方法时需使用 super( )，下面将使用另一类的类了解这个概念。

程序实例 ch12_14.py：这是一个衍生类调用基类方法的实例，首先建立一个 Animals 类，然后建立这个类的衍生类 Dogs，Dogs 类在初始化中会使用 super( ) 调用 Animals 类的初始化方法，可参考第 14 行，经过初始化处理后，mydog.name 将由“lily”变为“My pet lily”。

```
1  # ch12_14.py
2  class Animals():
3      """Animals类, 这是基类 """
4      def __init__(self, animal_name, animal_age ):
5          self.name = animal_name # 记录动物名称
6          self.age = animal_age   # 记录动物年龄
7
8      def run(self):              # 输出动物 is running
9          print(self.name.title(), " is running")
10
11 class Dogs(Animals):
12     """Dogs类, 这是Animal的衍生类 """
13     def __init__(self, dog_name, dog_age):
14         super().__init__('My pet ' + dog_name.title(), dog_age)
15
16 mycat = Animals('lucy', 5)      # 建立Animals对象以及测试
17 print(mycat.name.title(), ' is ', mycat.age, " years old.")
18 mycat.run()
19
20 mydog = Dogs('lily', 6)         # 建立Dogs对象以及测试
21 print(mydog.name.title(), ' is ', mydog.age, " years old.")
22 mydog.run()
```

执行结果

```
===================== RESTART: D:\Python\ch12\ch12_14.py =====================
Lucy  is  5  years old.
Lucy  is running
My Pet Lily  is  6  years old.
My Pet Lily  is running
```

12-3-6　衍生类有自己的方法

面向对象设计很重要的一环是衍生类有自己的方法，

程序实例 ch12_14_1.py：扩充 ch12_14.py，让 Dogs 类有自己的方法 sleeping( )。

```
1  # ch12_14_1.py
2  class Animals():
3      """Animals类, 这是基类 """
4      def __init__(self, animal_name, animal_age ):
5          self.name = animal_name # 记录动物名称
6          self.age = animal_age   # 记录动物年龄
7
8      def run(self):              # 输出动物 is running
9          print(self.name.title(), " is running")
10
11 class Dogs(Animals):
12     """Dogs类, 这是Animal的衍生类 """
13     def __init__(self, dog_name, dog_age):
14         super().__init__('My pet ' + dog_name.title(), dog_age)
15     def sleeping(self):
16         print("My pet", "is sleeping")
17
18 mycat = Animals('lucy', 5)      # 建立Animals对象以及测试
19 print(mycat.name.title(), ' is ', mycat.age, " years old.")
20 mycat.run()
21
22 mydog = Dogs('lily', 6)         # 建立Dogs对象以及测试
23 print(mydog.name.title(), ' is ', mydog.age, " years old.")
24 mydog.run()
25 mydog.sleeping()
```

执行结果

```
==================== RESTART: D:/Python/ch12/ch12_14_1.py ====================
Lucy  is  5  years old.
Lucy  is running
My Pet Lily  is  6  years old.
My Pet Lily  is running
My pet is sleeping
```

上述 Dogs 子类有一个自己的方法 sleep( )，第 25 行则是调用自己的子方法。

12-3-7 “三代同堂”的类与取得基类的属性 super( )

在继承的概念里，也可以使用 Python 的 super( ) 方法取得基类的属性，这对于设计“三代同堂”的类是很重要的。

下面是一个“三代同堂”的程序，在这个程序中有祖父（Grandfather）类，它的子类是父亲（Father）类，父亲类的子类是 Ivan 类。其实 Ivan 要取得父亲类的属性很容易，可是要取得祖父类的属性时就会碰上困难，解决方式是在 Father 类与 Ivan 类的 __init__( ) 方法中增加下列设置。

```
super( ).__init__( )            # 将父类的属性复制
```

这样 Ivan 就可以取得祖父（Grandfather）类的属性了。

程序实例 ch12_15.py：这个程序会建立一个 Ivan 类的对象 ivan，然后分别调用 Father 类和 Grandfather 类的方法打印信息，接着分别取得 Father 类和 Grandfather 类的属性。

```
1  # ch12_15
2  class Grandfather():
3      """ 定义祖父的资产 """
4      def __init__(self):
5          self.grandfathermoney = 10000
6      def get_info1(self):
7          print("Grandfather's information")
8
```

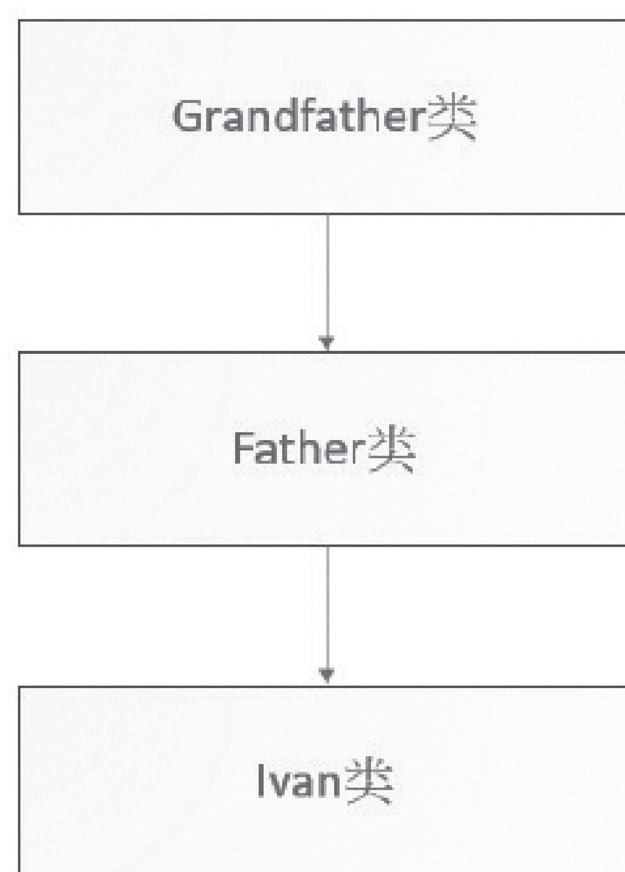
```
 9  class Father(Grandfather):        # 父类是Grandfather
10      """ 定义父亲的资产 """
11      def __init__(self):
12          self.fathermoney = 8000
13          super().__init__()
14      def get_info2(self):
15          print("Father's information")
16
17  class Ivan(Father):               # 父类是Father
18      """ 定义Ivan的资产 """
19      def __init__(self):
20          self.ivanmoney = 3000
21          super().__init__()
22      def get_info3(self):
23          print("Ivan's information")
24      def get_money(self):          # 取得资产明细
25          print("\nIvan资产: ", self.ivanmoney,
26                "\n父亲资产: ", self.fathermoney,
27                "\n祖父资产: ", self.grandfathermoney)
28
29  ivan = Ivan()
30  ivan.get_info3()                  # 从Ivan中获得
31  ivan.get_info2()                  # 流程 Ivan -> Father
32  ivan.get_info1()                  # 流程 Ivan -> Father -> Grandtather
33  ivan.get_money()                  # 取得资产明细
```

执行结果

```
==================== RESTART: D:\Python\ch12\ch12_15.py ====================
Ivan's information
Father's information
Grandfather's information

Ivan资产:  3000
父亲资产:  8000
祖父资产:  10000
```

上述程序各类的相互关系如下。



12-3-8　兄弟类属性的取得

假设有一个父亲（Father）类，这个父亲类有两个儿子分别是 Ivan 类和 Ira 类，如果 Ivan 类想取得 Ira 类的属性 iramoney，可以使用下列方法。

```
Ira( ).iramoney                # Ivan 取得 Ira 的属性 iramoney
```

程序实例 ch12_16.py：设计 3 个类，Father 类是 Ivan 和 Ira 类的父类，所以 Ivan 和 Ira 算是兄弟类，这个程序可以从 Ivan 类分别读取 Father 和 Ira 类的资产属性。这个程序中最重要的是第 21 行，

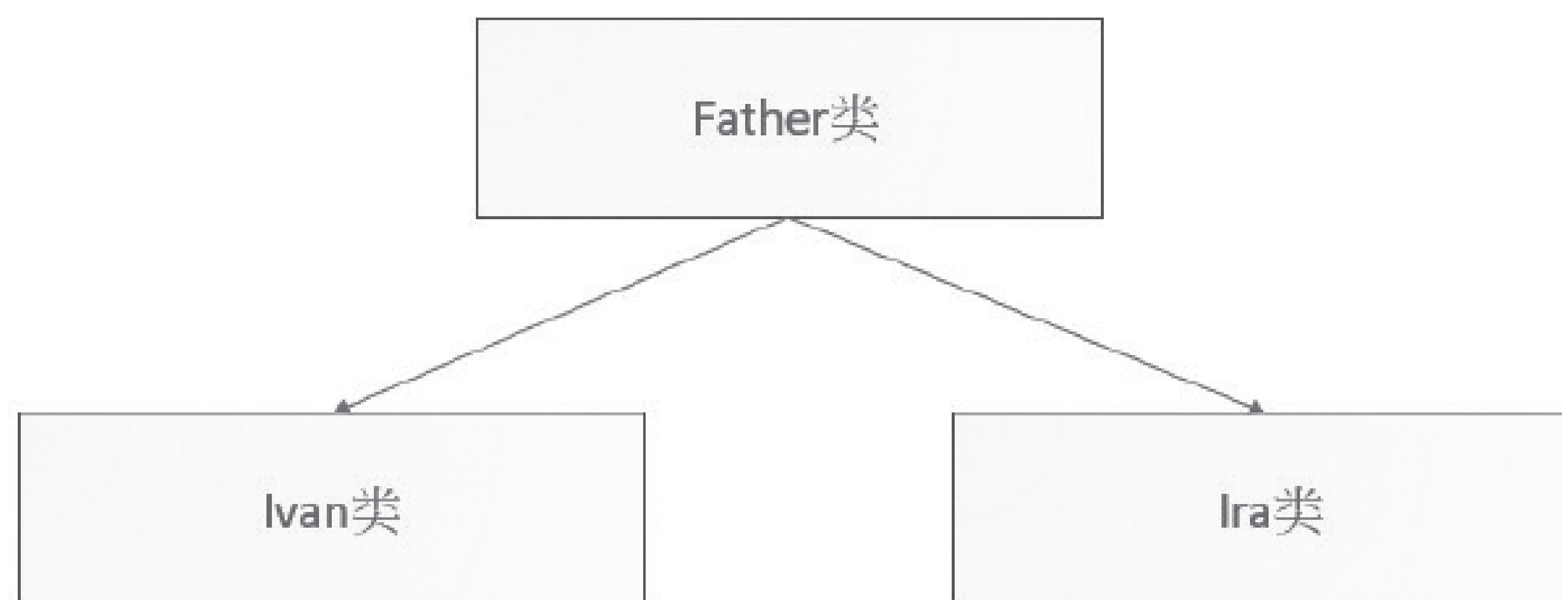
请留意取得 Ira 属性的写法。

```
1  # ch12_16.py
2  class Father():
3      """ 定义父亲的资产 """
4      def __init__(self):
5          self.fathermoney = 10000
6
7  class Ira(Father):                                # 父类是Father
8      """ 定义Ira的资产 """
9      def __init__(self):
10         self.iramoney = 8000
11         super().__init__()
12
13 class Ivan(Father):                               # 父类是Father
14     """ 定义Ivan的资产 """
15     def __init__(self):
16         self.ivanmoney = 3000
17         super().__init__()
18     def get_money(self):                          # 取得资产明细
19         print("Ivan资产: ", self.ivanmoney,
20               "\n父亲资产: ", self.fathermoney,
21               "\nIra资产 : ", Ira().iramoney)     # 注意写法
22
23 ivan = Ivan()
24 ivan.get_money()                                  # 取得资产明细
```

执行结果

```
==================== RESTART: D:\Python\ch12\ch12_16.py ====================
Ivan资产:  3000
父亲资产:  10000
Ira资产 :  8000
```

上述程序各类的相互关系如下。



12-3-9 认识 Python 类方法的 self 参数

如果读者懂 Java 可以知道类的方法没有 self 参数，本节将用一个简单的实例，讲解 self 参数的概念。

程序实例 ch12_16_1.py：建立类对象与调用类方法。

```
1  # ch12_16_1.py
2  class Person():
3      def interest(self):
4          print("Smiling is my interest")
5
6  hung = Person()
7  hung.interest()
```

执行结果

```
==================== RESTART: D:/Python/ch12/ch12_16_1.py ====================
Smiling is my interest
```

其实上述第 7 行相当于将 hung 当作 self 参数，然后传递给 Person 类的 interest( ) 方法。甚至也可以用下列方式，获得相同的输出。

```
>>> Person.interest(hung)
Smiling is my interest
```

上述语句只是有趣，不建议如此。

12-4 多态

在 12-3-4 节已经有说明基类与衍生类有相同方法名称的实例，其实那就是本节将要说明的**多态**（polymorphism）的基本概念，但是在**多态**的概念中是不局限在必须有父子关系的类。

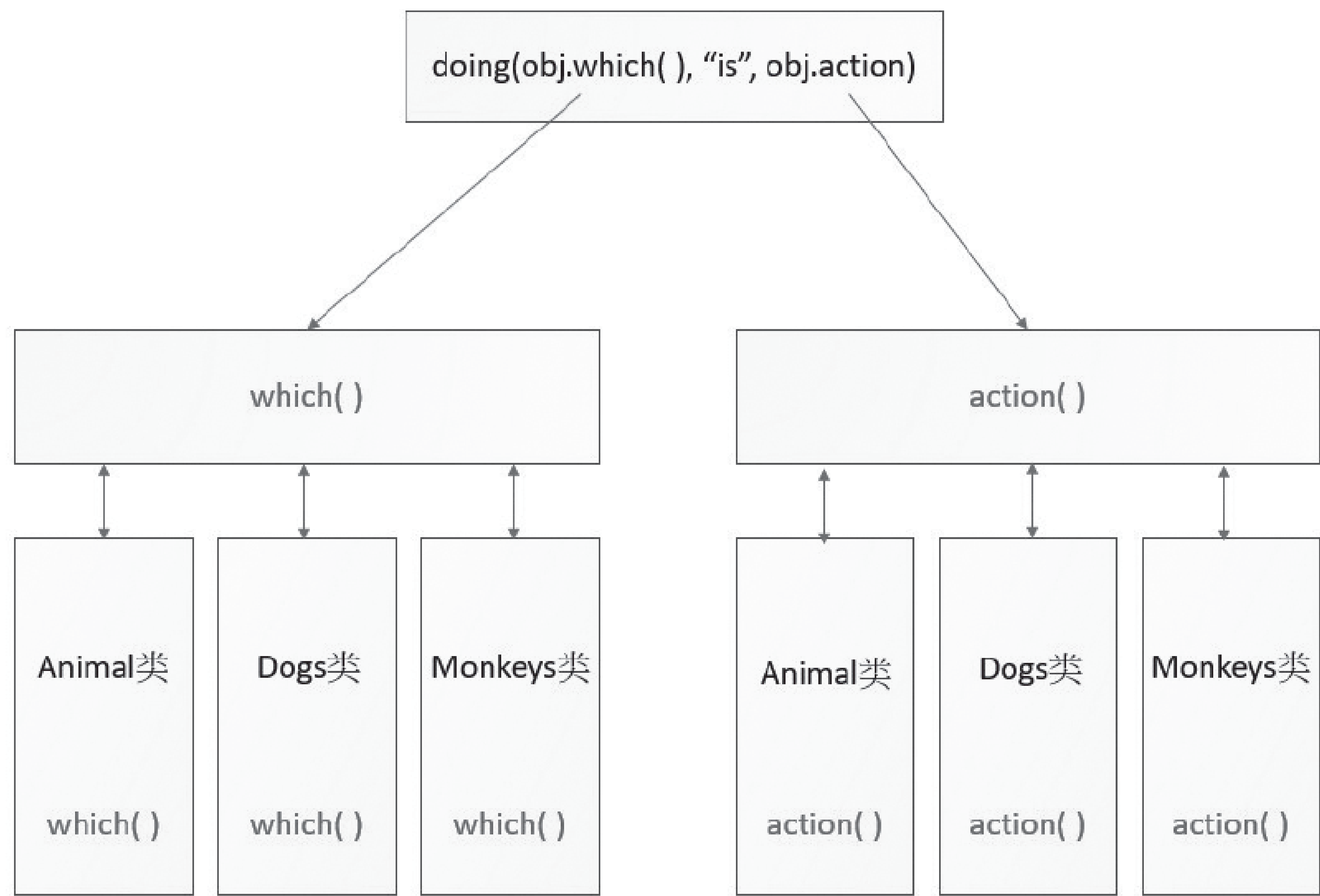
程序实例 ch12_17.py：这个程序有 3 个类，Animals 类是基类，Dogs 类是 Animals 类的衍生类，基于继承的特性所以两个类都有 which( ) 和 action( ) 方法，另外设计了一个与上述无关的类 Monkeys，这个类也有 which( ) 和 action( ) 方法，然后程序分别调用 which( ) 和 action( ) 方法，程序会由对象类判断应该使用哪一个方法响应程序。

```
1   # ch12_17.py
2   class Animals():
3       """Animals类, 这是基类 """
4       def __init__(self, animal_name):
5           self.name = animal_name          # 记录动物名称
6       def which(self):                     # 返回动物名称
7           return 'My pet ' + self.name.title()
8       def action(self):                    # 动物的行为
9           return ' sleeping'
10
11  class Dogs(Animals):
12      """Dogs类, 这是Animal的衍生类 """
13      def __init__(self, dog_name):        # 记录动物名称
14          super().__init__(dog_name.title())
15      def action(self):                    # 动物的行为
16          return ' running in the street'
17
18  class Monkeys():
19      """猴子类, 这是其他类 """
20      def __init__(self, monkey_name):     # 记录动物名称
21          self.name = 'My monkey ' + monkey_name.title()
22      def which(self):                     # 返回动物名称
23          return self.name
24      def action(self):                    # 动物的行为
25          return ' running in the forest'
26
27  def doing(obj):                          # 列出动物的行为
28      print(obj.which(), "is", obj.action())
29
30  my_cat = Animals('lucy')                 # Animals对象
31  doing(my_cat)
32  my_dog = Dogs('gimi')                    # Dogs对象
33  doing(my_dog)
34  my_monkey = Monkeys('taylor')            # Monkeys 对象
35  doing(my_monkey)
```

执行结果

```
===================== RESTART: D:\Python\ch12\ch12_17.py =====================
My pet Lucy is  sleeping
My pet Gimi is  running in the street
My monkey Taylor is  running in the forest
```

上述程序各类的相互关系如下。



对上述程序而言，第 30 行的 my_cat 是 Animal 类对象，所以在第 31 行此对象会触发 Animal 类的 which( ) 和 action( ) 方法。第 32 行的 my_dog 是 Dogs 类对象，所以在第 32 行此对象会触发 Dogs 类的 which( ) 和 action( ) 方法。第 34 行的 my_monkey 是 Monkeys 类对象，所以在第 35 行此对象会触发 Monkeys 类的 which( ) 和 action( ) 方法。

12-5 多重继承

12-5-1 基本概念

在面向对象的程序设计中，也常会发生一个类继承多个类的应用，此时子类也同时继承了多个类的方法。在这个时候，读者应该了解发生多个父类拥有相同名称的方法时，应该先执行哪一个父类的方法。在程序中可用下列语法代表继承多个类。

```
class 类名称(父类1, 父类2, …, 父类n):
    类内容
```

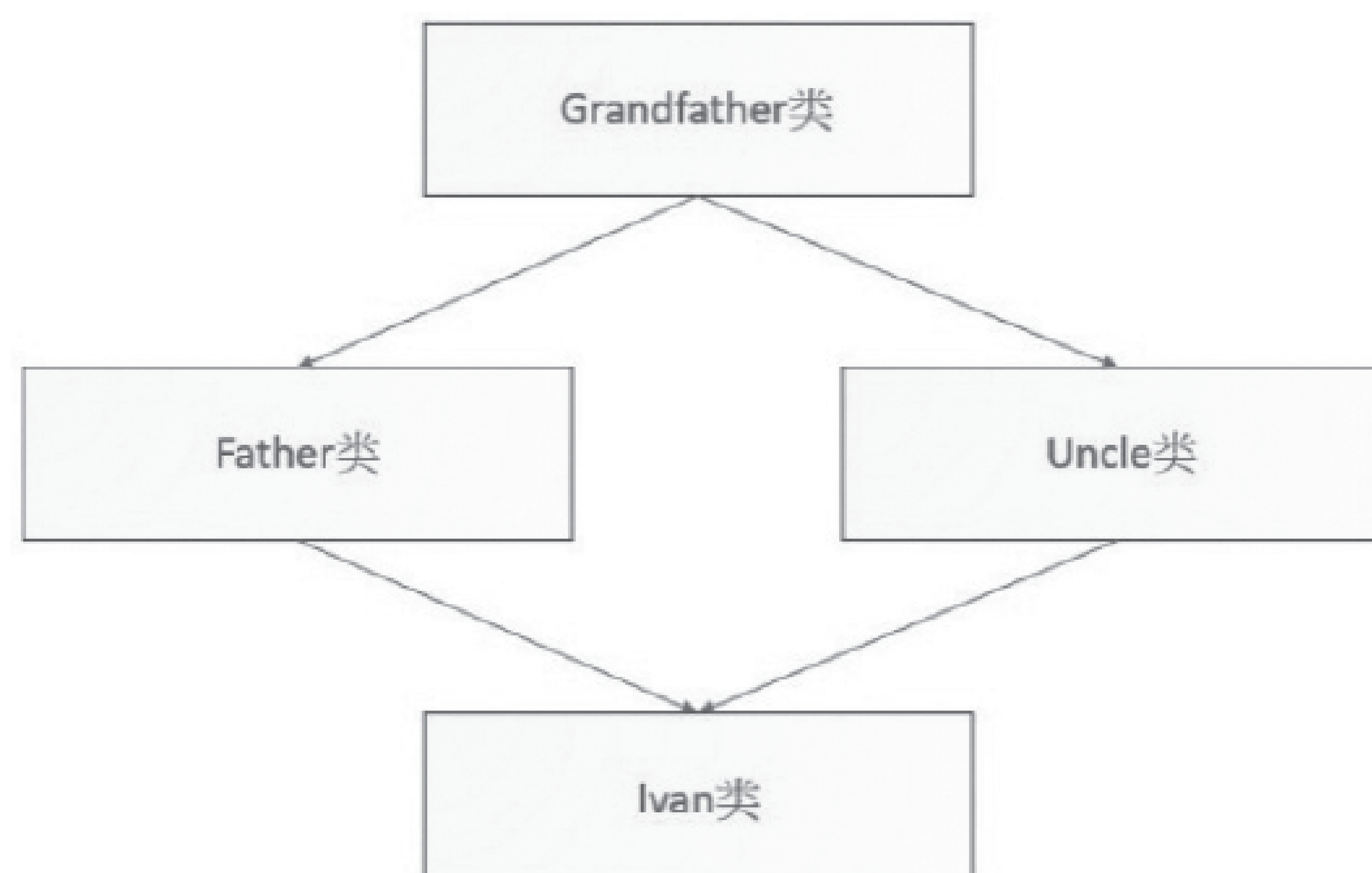
程序实例 ch12_18.py：这个程序 Ivan 类继承了 Father 和 Uncle 类，Grandfather 类则是 Father 和 Uncle 类的父类。在这个程序中只设置一个 Ivan 类的对象 ivan，然后由这个类分别调用 action3( )、action2( ) 和 action1( )，其中，Father 和 Uncle 类同时拥有 action2( ) 方法，读者可以观察最后是执行哪一个 action2( ) 方法。

```
1  # ch12_18.py
2  class Grandfather():
3      """ 定义祖父类 """
4      def action1(self):
5          print("Grandfather")
6
7  class Father(Grandfather):
8      """ 定义父亲类 """
9      def action2(self):         # 定义action2()
10         print("Father")
11
12 class Uncle(Grandfather):
13     """ 定义叔父类 """
14     def action2(self):         # 定义action2()
15         print("Uncle")
16
17 class Ivan(Father, Uncle):
18     """ 定义Ivan类 """
19     def action3(self):
20         print("Ivan")
21
22 ivan = Ivan()
23 ivan.action3()                 # 顺序 Ivan
24 ivan.action2()                 # 顺序 Ivan -> Father
25 ivan.action1()                 # 顺序 Ivan -> Father -> Grandfather
```

执行结果

```
==================== RESTART: D:\Python\ch12\ch12_18.py ====================
Ivan
Father
Grandfather
```

上述程序各类的相互关系如下。



程序实例 ch12_19.py：这个程序基本上是重新设计 ch12_18.py，主要是 Father 和 Uncle 类的方法名称是不一样的，Father 类是 action3( ) 和 Uncle 类是 action2( )，这个程序在建立 Ivan 类的 ivan 对象后，会分别启动各类的 actionX( ) 方法。

```
1  # ch12_19.py
2  class Grandfather():
3      """ 定义祖父类 """
4      def action1(self):
5          print("Grandfather")
6
7  class Father(Grandfather):
8      """ 定义父亲类 """
9      def action3(self):         # 定义action3()
10         print("Father")
11
```

```
12  class Uncle(Grandfather):
13      """ 定义叔父类 """
14      def action2(self):          # 定义action2()
15          print("Uncle")
16
17  class Ivan(Father, Uncle):
18      """ 定义Ivan类 """
19      def action4(self):
20          print("Ivan")
21
22  ivan = Ivan()
23  ivan.action4()                  # 顺序 Ivan
24  ivan.action3()                  # 顺序 Ivan -> Father
25  ivan.action2()                  # 顺序 Ivan -> Father -> Uncle
26  ivan.action1()                  # 顺序 Ivan -> Father -> Uncle -> Grandfather
```

执行结果

```
==================== RESTART: D:\Python\ch12\ch12_19.py ====================
Ivan
Father
Uncle
Grandfather
```

12-5-2 super( ) 应用于多重继承的问题

我们知道 super( ) 可以继承父类的方法，下面先看看可能产生的问题。

程序实例 ch12_19_1.py：super( ) 应用于多重继承的问题。

```
1   # ch12_19_1.py
2   class A():
3       def __init__(self):
4           print('class A')
5
6   class B():
7       def __init__(self):
8           print('class B')
9
10  class C(A,B):
11      def __init__(self):
12          super().__init__()
13          print('class C')
14
15  x = C()
```

执行结果

```
=================== RESTART: D:/Python/ch12/ch12_19_1.py ===================
class A
class C
```

上述第 10 行设置类 C 继承类 A 和 B，可是当我们设置对象 x 是类 C 的对象时，可以发现第 10 行 C 类的第 2 个参数 B 类没有被启动。其实 Python 使用 super( ) 的多重继承，在此算是协同作业（co-operative），我们必须在基类中也增加 super( ) 设置，才可以正常作业。

程序实例 ch12_19_2.py：重新设计 ch12_19_1.py，增加第 4 行和第 9 行，解决一般常见 super( ) 应用于多重继承的问题。

```
1  # ch12_19_2.py
2  class A():
3      def __init__(self):
4          super().__init__()
5          print('class A')
6
7  class B():
8      def __init__(self):
9          super().__init__()
10         print('class B')
11
12 class C(A,B):
13     def __init__(self):
14         super().__init__()
15         print('class C')
16
17 x = C()
```

执行结果

```
==================== RESTART: D:/Python/ch12/ch12_19_2.py ====================
class B
class A
class C
```

上述语句得到所有类的初始化方法（__init__( )）均被启动了，这个概念很重要，因为我们如果在初始化方法中想要子类继承所有父类的属性时，必须要使全部的父类均被启动，例如可以参考 ex12_9.py。

12-6 type 与 instance

一个大型程序可能是由许多人合作设计的，有时我们想了解某个对象变量的数据类型，或是所属类关系，可以使用本节所述的方法。

12-6-1 type( )

type( ) 函数先前已经使用许多次了，可以使用 type( ) 函数得到某一对象变量的类名称。

程序实例 ch12_20.py：列出类对象与对象内方法的数据类型。

```
1  # ch12_20.py
2  class Grandfather():
3      """ 定义祖父类 """
4      pass
5
6  class Father(Grandfather):
7      """ 定义父亲类 """
8      pass
9
10 class Ivan(Father):
11     """ 定义Ivan类 """
12     def fn(self):
13         pass
14
15 grandfather = Grandfather()
16 father = Father()
17 ivan = Ivan()
18 print("grandfather对象类型: ", type(grandfather))
19 print("father对象类型      : ", type(father))
20 print("ivan对象类型        : ", type(ivan))
21 print("ivan对象fn方法类型 : ", type(ivan.fn))
```

执行结果

```
===================== RESTART: D:\Python\ch12\ch12_20.py =====================
grandfather对象类型:  <class '__main__.Grandfather'>
father对象类型      :  <class '__main__.Father'>
ivan对象类型        :  <class '__main__.Ivan'>
ivan对象fn方法类型  :  <class 'method'>
```

由上述可以得到类的对象类型是 class，同时会列出“__main__. 类的名称”。如果是类内的方法同时也列出“method”方法。

12-6-2 isinstance( )

isinstance( ) 函数可以返回对象的类是否属于某一类，它包含两个参数，语法如下。

```
isinstance(对象, 类)            # 可返回 True 或 False
```

如果对象的类是属于第 2 个参数类或属于第 2 个参数的子类，则返回 True，否则返回 False。

程序实例 ch12_21.py：一系列 isinstance( ) 函数的测试。

```
1  # ch12_21.py
2  class Grandfather():
3      """ 定义祖父类 """
4      pass
5
6  class Father(Grandfather):
7      """ 定义父亲类 """
8      pass
9
10 class Ivan(Father):
11     """ 定义Ivan类 """
12     def fn(self):
13         pass
14
15 grandfa = Grandfather()
16 father = Father()
17 ivan = Ivan()
18 print("ivan属于Ivan类: ", isinstance(ivan, Ivan))
19 print("ivan属于Father类: ", isinstance(ivan, Father))
20 print("ivan属于GrandFather类: ", isinstance(ivan, Grandfather))
21 print("father属于Ivan类: ", isinstance(father, Ivan))
22 print("father属于Father类: ", isinstance(father, Father))
23 print("father属于Grandfather类: ", isinstance(father, Grandfather))
24 print("grandfa属于Ivan类: ", isinstance(grandfa, Ivan))
25 print("grandfa属于Father类: ", isinstance(grandfa, Father))
26 print("grandfa属于Grandfather类: ", isinstance(grandfa, Grandfather))
```

执行结果

```
===================== RESTART: D:\Python\ch12\ch12_21.py =====================
ivan属于Ivan类:  True
ivan属于Father类:  True
ivan属于GrandFather类:  True
father属于Ivan类:  False
father属于Father类:  True
father属于Grandfather类:  True
grandfa属于Ivan类:  False
grandfa属于Father类:  False
grandfa属于Grandfather类:  True
```

12-7 特殊属性

其实设计或是看到别人设计的 Python 程序时，若是看到 __xx__ 类的字符串就要特别留意了，这些大多数是特殊属性或方法，本书将简要说明几个重要且常见的。

12-7-1　文件字符串 __doc__

在 11-6-1 节已经有一些说明，本节将以程序实例解说。文件字符串的英文原意是 docstring，Python 鼓励程序设计师在设计函数或类时，尽量为函数或类增加文件的注释，未来可以使用 __doc__ 特殊属性列出此文件注释。

程序实例 ch12_22.py：将文件注释应用于函数。

```
1  # ch12_22.py
2  def getMax(x, y):
3      '''文件字符串实例
4  建议x, y是整数
5  这个函数将返回较大值'''
6      if int(x) > int(y):
7          return x
8      else:
9          return y
10
11 print(getMax(2, 3))          # 打印较大值
12 print(getMax.__doc__)        # 打印文件字符串docstring
```

执行结果

```
===================== RESTART: D:\Python\ch12\ch12_22.py =====================
3
文件字符串实例
建议x, y是整数
这个函数将传回较大值
```

程序实例 ch12_23.py：将文件注释应用于类与类内的方法。

```
1  # ch12_23.py
2  class Myclass:
3      '''文件字符串实例
4  Myclass类的应用'''
5      def __init__(self, x):
6          self.x = x
7      def printMe(self):
8          '''文本文件字符串实例
9  Myclass类内printMe方法的应用'''
10         print("Hi", self.x)
11
12 data = Myclass(100)
13 data.printMe()
14 print(data.__doc__)            # 打印Myclass文件字符串docstring
15 print(data.printMe.__doc__)    # 打印printMe文件字符串docstring
```

执行结果

```
===================== RESTART: D:\Python\ch12\ch12_23.py =====================
Hi 100
文件字符串实例
Myclass类的应用
文本文件字符串实例
Myclass类内printMe方法的应用
```

了解以上概念后，如果读者看到有一个程序代码如下：

```
>>> x = 'abc'
>>> print(x.__doc__)
str(object='') -> str
str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or
errors is specified, then the object must expose a data buffer
that will be decoded using the given encoding and error handler.
Otherwise, returns the result of object.__str__() (if defined)
or repr(object).
encoding defaults to sys.getdefaultencoding().
errors defaults to 'strict'.
>>>
```

以上只是列出 Python 系统内部有关字符串的 docstring。

12-7-2 __name__ 属性

如果你是 Python 程序设计师，常在网络上看别人写的程序，一定会经常在程序末端看到下列语句。

```
if __name__ == '__main__':
doSomething( )
```

初学 Python 时，笔者照上述语句编写，程序一定可以执行，当时不明白意思，觉得应该要告诉读者。如果上述程序是自己执行，那么 __name__ 就一定是 __main__。

程序实例 ch12_24.py：一个程序只有一行，就是打印 __name__。

```
1  # ch12_24.py
2  print('ch12_24.py module name = ', __name__)
```

执行结果

```
==================== RESTART: D:\Python\ch12\ch12_24.py ====================
ch12_24.py module name =  __main__
```

经过上述实例我们知道，如果程序是自己执行时，__name__ 就是 __main__。所以下列程序实例可以列出结果。

程序实例 ch12_25.py：__name__ == __main__ 的应用。

```
1  # ch12_25.py
2  def myFun():
3      print("__name__ == __main__")
4  if __name__ == '__main__':
5      myFun()
```

执行结果

```
==================== RESTART: D:\Python\ch12\ch12_25.py ====================
__name__ == __main__
```

如果 ch12_24.py 是被 import 到另一个程序时，则 __name__ 是本身的文件名。第 13 章会介绍关于 import 的知识，它的用途是将模块导入，方便程序调用。

程序实例 ch12_26.py：这个程序用 import 导入 ch12_24.py，结果 __name__ 变成了 ch12_24。

```
1  # ch12_26.py
2  import ch12_24
```

执行结果

```
===================== RESTART: D:\Python\ch12\ch12_26.py =====================
ch12_24.py module name =  ch12_24
```

程序实例 ch12_27.py：这个程序用 import 导入 ch12_25.py，由于 __name__ 已经不再是 __main__，所以程序没有任何输出。

```
1  # ch12_27.py
2  import ch12_25
```

执行结果

```
===================== RESTART: D:\Python\ch12\ch12_27.py =====================
```

所以 __name__ 可以判别这个程序是自己执行还是被其他程序 import 导入当成模块使用的。

12-8 类的特殊方法

12-8-1 __str__( ) 方法

这是类的特殊方法，可以协助返回易读取的字符串。

程序实例 ch12_28.py：在没有定义 __str__( ) 方法的情况下，列出类的对象。

```
1  # ch12_28.py
2  class Name:
3      def __init__(self, name):
4          self.name = name
5
6  a = Name('Hung')
7  print(a)
```

执行结果

```
===================== RESTART: D:\Python\ch12\ch12_28.py =====================
<__main__.Name object at 0x03624830>
```

上述语句在没有定义 __str__( ) 方法的情况下，获得了一个不太容易阅读的结果。

程序实例 ch12_29.py：在定义 __str__( ) 方法的情况下，重新设计上一个程序。

```
1  # ch12_29.py
2  class Name:
3      def __init__(self, name):
4          self.name = name
5      def __str__(self):
6          return '%s' % self.name
7
8  a = Name('Hung')
9  print(a)
```

执行结果

```
===================== RESTART: D:\Python\ch12\ch12_29.py =====================
Hung
```

上述语句定义了 __str__( ) 方法后，就得到一个适合阅读的结果了。对于程序 ch12_29.py 而言，如果在 Python Shell 窗口中输入 a，将同样获得不容易阅读的结果。

```
===================== RESTART: D:\Python\ch12\ch12_29.py =====================
Hung
>>> a
<__main__.Name object at 0x04204850>
```

12-8-2 __repr__( ) 方法

如果只是在 Python Shell 窗口中读入类变量 a，系统是调用 __repr__( ) 方法做响应，为了要获得容易阅读的结果，也需要定义此方法。

程序实例 ch12_30.py：定义 __repr__( ) 方法，其实此方法的内容与 __str__( ) 相同，所以可以用等号取代。

```
1   # ch12_30.py
2   class Name:
3       def __init__(self, name):
4           self.name = name
5       def __str__(self):
6           return '%s' % self.name
7       __repr__ = __str__
8
9   a = Name('Hung')
10  print(a)
```

执行结果

```
===================== RESTART: D:\Python\ch12\ch12_30.py =====================
Hung
>>> a
Hung
```

12-8-3 __iter__( ) 方法

建立类的时候也可以将类定义成一个迭代对象，类似 list 或 tuple，供 for … in 循环内使用，这时类需设计 next( ) 方法，取得下一个值，直到达到结束条件，可以使用 raise StopIteration（第 15 章

会介绍，raise）终止进程。

程序实例 ch12_31.py：Fib 序列数的设计。

```
1  # ch12_31.py
2  class Fib():
3      def __init__(self, max):
4          self.max = max
5
6      def __iter__(self):
7          self.a = 0
8          self.b = 1
9          return self
10
11     def __next__(self):
12         fib = self.a
13         if fib > self.max:
14             raise StopIteration
15         self.a, self.b = self.b, self.a + self.b
16         return fib
17 for i in Fib(100):
18     print(i)
```

执行结果

```
===================== RESTART: D:\Python\ch12\ch12_31.py =====================
0
1
1
2
3
5
8
13
21
34
55
89
```

12-8-4　__eq__( ) 方法

假设我们想要了解两个字符串或其他内容是否相同，依照我们的知识可以使用下列方式设计。

程序实例 ch12_32.py：设计检查字符串是否相等。

```
1  # ch12_32.py
2  class City():
3      def __init__(self, name):
4          self.name = name
5      def equals(self, city2):
6          return self.name.upper() == city2.name.upper()
7
8  one = City("Taipei")
9  two = City("taipei")
10 three = City("myhome")
11 print(one.equals(two))
12 print(one.equals(three))
```

执行结果

```
===================== RESTART: D:/Python/ch12/ch12_32.py =====================
True
False
```

现在将 equals( ) 方法改为 __eq( )__，可以参考下列实例。

程序实例 ch12_33.py：使用 __eq( )__ 取代 equals( ) 方法，可以得到和 ch12_32.py 相同的结果。

```
1  # ch12_33.py
2  class City():
3      def __init__(self, name):
4          self.name = name
5      def __eq__(self, city2):
6          return self.name.upper() == city2.name.upper()
7
8  one = City("Taipei")
9  two = City("taipei")
10 three = City("myhome")
11 print(one == two)
12 print(one == three)
```

执行结果 与 ch12_32.py 相同。

上述是类的特殊方法，主要是了解内容是否相同，下面是拥有这类特点的其他系统方法。

逻辑方法	说明
__eq__(self, other)	self == other # 等于
__ne__(self, other)	self != other # 不等于
__lt__(self, other)	self < other # 小于
__gt__(self, other)	self > other # 大于
__le__(self, other)	self <= other # 小于或等于
__ge__(self, other)	self >= other # 大于或等于

数学方法	说明
__add__(self, other)	self + other # 加法
__sub__(self, other)	self – other # 减法
__mul__(self, other)	self * other # 乘法
__floordiv__(self, other)	self / /other # 整数除法
__truediv__(self, other)	self / other # 除法
__mod__(self, other)	self % other # 余数
__pow__(self, other)	self ** other # 次方

12-9 专题——几何数据的应用

程序实例 ch12_34.py：设计一个 Geometric 类，这个类主要是设置 color 是 Green。另外设计一个 Circle 类，这个类有 getRadius( ) 方法可以获得半径，setRadius( ) 方法可以设置半径，getDiameter( ) 方法可以取得直径，getPerimeter( ) 方法可以取得圆周长，getArea( ) 方法可以取得面积，getColor( ) 方法可以取得颜色。

```
1  # ch12_34.py
2  class Geometric():
3      def __init__(self):
4          self.color = "Green"
5  class Circle(Geometric):
6      def __init__(self,radius):
7          super().__init__()
8          self.PI = 3.14159
9          self.radius = radius
10     def getRadius(self):
11         return self.radius
12     def setRadius(self,radius):
13         self.radius = radius
14     def getDiameter(self):
15         return self.radius * 2
16     def getPerimeter(self):
17         return self.radius * 2 * self.PI
18     def getArea(self):
19         return self.PI * (self.radius ** 2)
20     def getColor(self):
21         return color
22
23 A = Circle(5)
24 print("圆形的颜色 : ", A.color)
25 print("圆形的半径 : ", A.getRadius())
26 print("圆形的直径 : ", A.getDiameter())
27 print("圆形的圆周 : ", A.getPerimeter())
28 print("圆形的面积 : ", A.getArea())
29 A.setRadius(10)
30 print("圆形的直径 : ", A.getDiameter())
```

执行结果

```
===================== RESTART: D:\Python\ch12\ch12_34.py =====================
圆形的颜色 :  Green
圆形的半径 :  5
圆形的直径 :  10
圆形的圆周 :  31.4159
圆形的面积 :  78.53975
圆形的直径 :  20
```

习题

1. 设计一个类 Myschool，这个类包含属性 title，也有一个 departments( ) 方法，属性内容如下。(12-1 节)

title = " 明志科大 "

departments( ) 方法则是返回列表 [" 机械 ", " 电机 ", " 化工 "]

读者需声明一个 Myschool 对象，然后依下列方式打印信息。

```
====================== RESTART: D:\Python\ex\ex12_1.py ======================
明志科大
机械
电机
化工
```

2. 设计一个类 Myschool，这个类包含属性 name 和 score，也有一个 msg( ) 方法，程序设置 Myschool 对象时需传递两个参数，下面是示范设置方式。(12-1 节)

hung = Myschool('kevin', 80)

这个类的方法主要是可以输出问候语和成绩，请留意英文名字第一个输出字母是大写。

```
====================== RESTART: D:\Python\ex\ex12_2.py ======================
Hi!Kevin你的成绩是80分
```

3. 请扩充习题 1，增加初始化 schoolname 属性，schoolname 内容是 'Python School'，请设计 msg( ) 方法输出第一行是 title，第二行才是原先的输出。(12-1 节)

```
===================== RESTART: D:\Python\ex\ex12_3.py =====================
Python School
Hi!Kevin你的成绩是80分
```

4. 请利用 ch12_9.py 的类，同时修改部分内容，在程序部分执行下列工作 :(12-2 节)

（1）存款 5000 元 ;

（2）提款 3000 元 ;

（3）存款 1500 元 ;

（4）兑换美金外币 100 美元（记住 : 汇率是要增加手续费用 1%）;

（5）列出剩余金额。

请列出上述每次的执行结果账单。

```
===================== RESTART: D:\Python\ex\ex12_4.py =====================
存款  5000  完成
Hung  目前余额:  5000
提款  3000  完成
Hung  目前余额:  2000
存款  1500  完成
Hung  目前余额:  3500
兑换100美元
提款  3030  完成
Hung  目前余额:  470
```

5. 请扩充 ch12_13.py，增加 Banks 子类北投（Beitou）分行，北投分行内容可以参照士林分行，程序末端增加北投分行类对象（可参考 43 行），然后打印银行名称（可参考 44 行）。(12-3 节)

```
===================== RESTART: D:/Python/ex/ex12_5.py =====================
James's banks =  Taipei Bank
Hung's banks  =  Taipei Bank - Shilin Branch
Kevin's banks =  Taipei Bank - Shilin Branch
```

6. 请扩充 ch12_14.py，为 Animals 类增加 Birds 子类，这个子类有自己的 run( ) 方法，输出方式可以比照第 9 行，但是字符串是“ is flying.”。请为这个程序增加类似 20 ～ 22 行的工作，但是将对象类设为 Birds。(12-3 节)

```
===================== RESTART: D:/Python/ex/ex12_6.py =====================
Lucy  is  5  years old.
Lucy  is running
My Pet Lily  is  6  years old.
My Pet Lily  is running
My Pet Cici  is  8  years old.
My Pet Cici is flying
```

7. 请适度修订 ch12_16.py，将第 23 行对象改为 :(12-3 节)

```
ira = Ira( )
```

第 24 行也需修改，在 Ira 类内增加设计方法可以调用 Ivan 类的 get_money( ) 方法，然后输出结果。

```
===================== RESTART: D:\Python\ex\ex12_7.py =====================
Ira资产:  8000
父亲资产:  10000
Ivan资产 :  3000
```

8. 请扩充 ch12_18.py，增加 Grandfather 类的子类 Aunt 类，这个类也是 Ivan 类的父类。请参考第 14 行建立 action2( ) 方法但是列出“Aunt”。在第 17 行 Ivan 类内的参数如下。(12-4 节)

Father, Uncle, Aunt　　　　　　　　--- ex12_8_1.py

请再设计两个程序参数分别如下。

Uncle, Aunt, Father　　　　　　　　--- ex12_8_2.py

Aunt, Father, Uncle　　　　　　　　--- ex12_8_3.py

同时列出结果。

```
==================== RESTART: D:/Python/ex/ex12_8_1.py ====================
Ivan
Father
Grandfather
```

```
==================== RESTART: D:/Python/ex/ex12_8_2.py ====================
Ivan
Uncle
Grandfather
```

```
==================== RESTART: D:/Python/ex/ex12_8_3.py ====================
Ivan
Aunt
Grandfather
```

9. 请扩充 ch12_15.py，增加 Grandmother 类，这是 Father 类的父类，她的资产是 20000，请参考 Grandfather 类建立 get_info4( ) 方法，同时在程序中扩充输出 Grandmother 的资产。(12-5 节)

```
===================== RESTART: D:\Python\ex\ex12_9.py =====================
Ivan's information
Father's information
Grandfather's information
Grandmother's information

Ivan资产:  3000
父亲资产:  8000
祖父资产:  10000
祖母资产:  20000
```

13

第 1 3 章

设计与应用模块

本章摘要

第 11 章介绍了函数（function），第 12 章介绍了类（class），其实在大型程序设计中，每个人可能只是负责一小部分的函数或类设计，为了可以让团队的其他人可以互相分享设计成果，最后每个人所负责的功能函数或类将存储在**模块**（module）中，然后供团队其他成员使用。在网络上或国外的技术文件中常可以看到有的文章将**模块**（module）称为**套件**（package）。

通常将模块分成以下 3 大类。

（1）自己程序建立的模块，本章 13-1 节至 13-4 节会做说明。

（2）Python 内建的模块，13-5 节至 13-10 节会有实例说明。例如，数学模块 math、随机数模块 random、文件处理模块 os、时间模块 time、系统模块 sys 等。可以使用下列网址查询所有 Python 内部模块：

http://docs.python.org/3/library

（3）外部模块，需使用 pip 安装，未来章节会在使用时说明，也可参考附录 B。

本章将讲解将自己所设计的函数或类存储成模块然后加以引用，最后也将讲解 Python 常用的内建模块。Python 最大的优势是资源免费，因此有许多公司使用它开发了许多功能强大的模块，这些模块称为外部模块或第三方模块，后面章节会逐步说明使用外部模块执行更多有意义的工作。

13-1　将自建的函数存储在模块中

一个大型程序一定是由许多的函数或类所组成的，为了让程序的工作可以分工以及增加程序的可读性，可以将所建的函数或类存储成模块（module）形式的独立文件，未来再加以调用。

13-1-1　准备工作

假设有一个程序内容是用于建立冰淇淋（ice cream）与饮料（drink），如下所示。

程序实例 ch13_1.py：这个程序基本上是扩充 ch11_23.py，再增加建立饮料的函数 make_drink( )。

```
1  # ch13_1.py
2  def make_icecream(*toppings):
3      # 列出制作冰淇淋的配料
4      print("这个冰淇淋所加配料如下")
5      for topping in toppings:
6          print("--- ", topping)
7
8  def make_drink(size, drink):
9      # 输入饮料规格与种类,然后输出饮料
10     print("所点饮料如下")
11     print("--- ", size.title())
12     print("--- ", drink.title())
13
14 make_icecream('草莓酱')
15 make_icecream('草莓酱', '葡萄干', '巧克力碎片')
16 make_drink('large', 'coke')
```

执行结果

```
==================== RESTART: D:\Python\ch13\ch13_1.py ====================
这个冰淇淋所加配料如下
---   草莓酱
这个冰淇淋所加配料如下
---   草莓酱
---   葡萄干
---   巧克力碎片
所点饮料如下
---   Large
---   Coke
```

假设我们会常常需要在其他程序中调用 make_icecream( ) 和 make_drink( )，此时可以考虑将这两个函数建立成模块，未来可以供其他程序调用。

13-1-2　建立函数内容的模块

模块的扩展名与 Python 程序文件一样是 py，对于程序实例 ch13_1.py 而言，可以只保留 make_icecream( ) 和 make_drink( )。

程序实例 makefood.py：使用 ch13_1.py 建立一个模块，此模块名称是 makefood.py。

```
1  # makefood.py
2  # 这是一个包含两个函数的模块(module)
3  def make_icecream(*toppings):
4      ''' 列出制作冰淇淋的配料 '''
5      print("这个冰淇淋所加配料如下")
6      for topping in toppings:
7          print("--- ", topping)
8
9  def make_drink(size, drink):
10     ''' 输入饮料规格与种类,然后输出饮料 '''
11     print("所点饮料如下")
12     print("--- ", size.title())
13     print("--- ", drink.title())
```

执行结果　由于这不是一般程序所以没有任何执行结果。

现在已经成功地建立模块 makefood.py 了。

13-2　应用自己建立的函数模块

有几种方法可以应用函数模块，下面将分成 6 节进行说明。

13-2-1　import 模块名称

要导入 13-1-2 节所建的模块，只要在程序内加上下列简单的语法即可。

```
import 模块名称                # 导入模块
```

若以 13-1-2 节的实例，只要在程序内加上下列简单的语句即可。

```
import makefood
```

程序中要引用模块的函数语法如下。

```
模块名称 . 函数名称            # 模块名称与函数名称间有小数点 "."
```

程序实例 ch13_2.py：实际导入模块 makefood.py 的应用。

```
1  # ch13_2.py
2  import makefood          # 导入模块makefood.py
3
4  makefood.make_icecream('草莓酱')
5  makefood.make_icecream('草莓酱', '葡萄干', '巧克力碎片')
6  makefood.make_drink('large', 'coke')
```

执行结果　与 ch13_1.py 相同。

13-2-2　导入模块内特定单一函数

如果只想导入模块内单一特定的函数，可以使用下列语法。

from 模块名称 import 函数名称

未来程序引用所导入的函数时可以省略模块名称。

程序实例 ch13_3.py：这个程序只导入 makefood.py 模块的 make_icecream( ) 函数，所以程序第 4 和 5 行执行没有问题，但是执行程序第 6 行时就会产生错误。

```
1  # ch13_3.py
2  from makefood import make_icecream  # 导入模块makefood.py的函数make_icecream
3
4  make_icecream('草莓酱')
5  make_icecream('草莓酱', '葡萄干', '巧克力碎片')
6  make_drink('large', 'coke')         # 因为没有导入此函数所以会产生错误
```

执行结果

```
==================== RESTART: D:\Python\ch13\ch13_3.py ====================
这个冰淇淋所加配料如下
---  草莓酱
这个冰淇淋所加配料如下
---  草莓酱
---  葡萄干
---  巧克力碎片
Traceback (most recent call last):
  File "D:\Python\ch13\ch13_3.py", line 6, in <module>
    make_drink('large', 'coke')         # 因为没有导入此函数所以会产生错误
NameError: name 'make_drink' is not defined
```

13-2-3　导入模块内多个函数

如果想导入模块内多个函数时，函数名称间需以逗号隔开，语法如下。

from 模块名称 import 函数名称 1, 函数名称 2, … , 函数名称 n

程序实例 ch13_4.py：重新设计 ch13_3.py，增加导入 make_drink( ) 函数。

```
1  # ch13_4.py
2  # 导入模块makefood.py的make_icecream和make_drink函数
3  from makefood import make_icecream, make_drink
4
5  make_icecream('草莓酱')
6  make_icecream('草莓酱', '葡萄干', '巧克力碎片')
7  make_drink('large', 'coke')
```

执行结果　与 ch13_1.py 相同。

13-2-4 导入模块所有函数

如果想导入模块内所有函数时，语法如下。

from 模块名称 import *

程序实例 ch13_5.py：导入模块所有函数的应用。

```
1  # ch13_5.py
2  from makefood import *          # 导入模块makefood.py所有函数
3
4  make_icecream('草莓酱')
5  make_icecream('草莓酱', '葡萄干', '巧克力碎片')
6  make_drink('large', 'coke')
```

执行结果 与 ch13_1.py 相同。

13-2-5 使用 as 给函数指定替代名称

有时候会碰上所设计程序的函数名称与模块内的函数名称相同，或是感觉模块的函数名称太长，此时可以自行给模块的函数名称一个**替代名称**，未来可以使用这个**替代名称**代替原先模块的名称。语法格式如下。

from 模块名称 import 函数名称 as **替代名称**

程序实例 ch13_6.py：使用替代名称 icecream 代替 make_icecream，重新设计 ch13_3.py。

```
1  # ch13_6.py
2  # 使用icecream替代make_icecream函数名称
3  from makefood import make_icecream  as icecream
4
5  icecream('草莓酱')
6  icecream('草莓酱', '葡萄干', '巧克力碎片')
```

执行结果

```
==================== RESTART: D:\Python\ch13\ch13_6.py ====================
这个冰淇淋所加配料如下
---   草莓酱
这个冰淇淋所加配料如下
---   草莓酱
---   葡萄干
---   巧克力碎片
```

13-2-6 使用 as 给模块指定替代名称

Python 也允许给模块替代名称，未来可以使用此替代名称导入模块，其语法格式如下。

import 模块名称 as 替代名称

程序实例 ch13_7.py：使用 m 当作模块替代名称，重新设计 ch13_2.py。

```
1  # ch13_7.py
2  import makefood as m           # 导入模块makefood.py的替代名称m
3
4  m.make_icecream('草莓酱')
5  m.make_icecream('草莓酱', '葡萄干', '巧克力碎片')
6  m.make_drink('large', 'coke')
```

执行结果 与 ch13_1.py 相同。

13-3 将自建的类存储在模块内

第 12 章介绍了类，当程序设计越来越复杂时，可能也会建立许多类，Python 也允许将所建立的类存储在模块内，这将是本节的重点。

13-3-1 准备工作

下面将使用第 12 章的程序实例，说明将类存储在模块中的方式。

程序实例 ch13_8.py：修改 ch12_13.py，简化 Banks 类，同时让程序有两个类，至于程序内容读者应该可以轻易了解。

```
1  # ch13_8.py
2  class Banks():
3      ''' 定义银行类 '''
4
5      def __init__(self, uname):                      # 初始化方法
6          self.__name = uname                         # 设置私有存款者名字
7          self.__balance = 0                          # 设置私有开户金额是0
8          self.__title = "Taipei Bank"                # 设置私有银行名称
9
10     def save_money(self, money):                    # 设计存款方法
11         self.__balance += money                     # 执行存款
12         print("存款 ", money, " 完成")               # 打印存款完成
13
14     def withdraw_money(self, money):                # 设计取款方法
15         self.__balance -= money                     # 执行取款
16         print("取款 ", money, " 完成")               # 打印取款完成
17
18     def get_balance(self):                          # 获得存款余额
19         print(self.__name.title(), " 目前余额: ", self.__balance)
20
21     def bank_title(self):                           # 获得银行名称
22         return self.__title
23
24 class Shilin_Banks(Banks):
25     ''' 定义士林分行 '''
26     def __init__(self, uname):
27         self.title = "Taipei Bank - Shilin Branch"  # 定义分行名称
28     def bank_title(self):                           # 获得银行名称
29         return self.title
30
31 jamesbank = Banks('James')                          # 定义Banks类对象
32 print("James's banks = ", jamesbank.bank_title())   # 打印银行名称
33 jamesbank.save_money(500)                           # 存钱
34 jamesbank.get_balance()                             # 列出存款金额
35 hungbank = Shilin_Banks('Hung')                     # 定义Shilin_Banks类对象
36 print("Hung's banks  = ", hungbank.bank_title())    # 打印银行名称
```

执行结果

```
===================== RESTART: D:\Python\ch13\ch13_8.py =====================
James's banks =  Taipei Bank
存款  500  完成
James  目前余额:  500
Hung's banks  =  Taipei Bank - Shilin Branch
```

13-3-2 建立类内容的模块

模块的扩展名与 Python 程序文件一样是 py，对于程序实例 ch13_8.py 而言，可以只保留 Banks 类和 Shilin_Banks 类。

程序实例 banks.py：使用 ch13_8.py 建立一个模块，此模块名称是 banks.py。

```
1  # banks.py
2  # 这是一个包含两 个类的模块(module)
3  class Banks():
4      ''' 定义银行类 '''
5      def __init__(self, uname):                  # 初始化方法
6          self.__name = uname                     # 设置私有存款者名字
7          self.__balance = 0                      # 设置私有开户金额是0
8          self.__title = "Taipei Bank"            # 设置私有银行名称
9
10     def save_money(self, money):                # 设计存款方法
11         self.__balance += money                 # 执行存款
12         print("存款 ", money, " 完成")           # 打印存款完成
13
14     def withdraw_money(self, money):            # 设计取款方法
15         self.__balance -= money                 # 执行取款
16         print("取款 ", money, " 完成")           # 打印取款完成
17
18     def get_balance(self):                      # 获得存款余额
19         print(self.__name.title(), " 目前余额: ", self.__balance)
20
21     def bank_title(self):                       # 获得银行名称
22         return self.__title
23
24 class Shilin_Banks(Banks):
25     ''' 定义士林分行 '''
26     def __init__(self, uname):
27         self.title = "Taipei Bank - Shilin Branch"  # 定义分行名称
28     def bank_title(self):                       # 获得银行名称
29         return self.title
```

执行结果 由于这不是程序所以没有任何执行结果。

现在已经成功地建立模块 banks.py 了。

13-4 应用自己建立的类模块

其实导入模块内的类与导入模块内的函数是一样的，下面将分成几节进行说明。

13-4-1 导入模块的单一类

方法与 13-2-2 节相同，语法格式如下。

```
from 模块名称 import 类名称
```

程序实例 ch13_9.py：使用导入模块方式，重新设计 ch13_8.py。由于这个程序只导入 Banks 类，所以此程序不执行原先的第 35 行和第 36 行。

```
1  # ch13_9.py
2  from banks import Banks                          # 导入banks模块的Banks类
3
4  jamesbank = Banks('James')                       # 定义Banks类对象
5  print("James's banks = ", jamesbank.bank_title())  # 打印银行名称
6  jamesbank.save_money(500)                        # 存钱
7  jamesbank.get_balance()                          # 列出存款金额
```

执行结果

```
===================== RESTART: D:\Python\ch13\ch13_9.py =====================
James's banks =  Taipei Bank
存款  500  完成
James  目前余额:  500
```

由执行结果读者应该体会，整个程序变得非常简洁了。

13-4-2　导入模块的多个类

与 13-2-3 节相同，如果模块内有多个类，也可以使用下列方式导入多个类，所导入的类名称间需以逗号隔开。

from 模块名称 import 类名称 1, 类名称 2, … , 类名称 n

程序实例 ch13_10.py：同时导入 Banks 类和 Shilin_Banks 类方式，重新设计 ch13_8.py。

```
1  # ch13_10.py
2  # 导入banks模块的Banks和Shilin_Banks类
3  from banks import Banks, Shilin_Banks
4
5  jamesbank = Banks('James')                          # 定义Banks类对象
6  print("James's banks = ", jamesbank.bank_title())  # 打印银行名称
7  jamesbank.save_money(500)                           # 存钱
8  jamesbank.get_balance()                             # 列出存款金额
9  hungbank = Shilin_Banks('Hung')                     # 定义Shilin_Banks类对象
10 print("Hung's banks  = ", hungbank.bank_title())   # 打印银行名称
```

执行结果　与 ch13_8.py 相同。

13-4-3　导入模块内所有类

与 13-2-4 节相同，如果想导入模块内所有类时，语法如下。

from 模块名称 import *

程序实例 ch13_11.py：使用导入模块所有类的方式重新设计 ch13_8.py。

```
1  # ch13_11.py
2  from banks import *                                 # 导入banks模块所有类
3
4  jamesbank = Banks('James')                          # 定义Banks类对象
5  print("James's banks = ", jamesbank.bank_title())  # 打印银行名称
6  jamesbank.save_money(500)                           # 存钱
7  jamesbank.get_balance()                             # 列出存款金额
8  hungbank = Shilin_Banks('Hung')                     # 定义Shilin_Banks类对象
9  print("Hung's banks  = ", hungbank.bank_title())   # 打印  # 打印银行名称
```

执行结果　与 ch13_8.py 相同。

13-4-4　import 模块名称

与 13-2-1 节相同，要导入 13-3-2 节所建的模块，只要在程序内加上下列简单的语法即可。

import 模块名称　　　　# 导入模块

若以 13-3-2 节的实例为例，只要在程序内加上下列简单的语法即可。

import banks

程序中要引用模块的类，语法如下。

模块名称 . 类名称　 **# 模块名称与类名称间有小数点 "."**

程序实例 ch13_12.py：使用 import 模块名称方式，重新设计 ch13_8.py，读者应该留意第 2、4 和 8 行的设计方式。

```
1  # ch13_12.py
2  import banks                                        # 导入banks模块
3
4  jamesbank = banks.Banks('James')                    # 定义Banks类对象
5  print("James's banks = ", jamesbank.bank_title())   # 打印银行名称
6  jamesbank.save_money(500)                           # 存钱
7  jamesbank.get_balance()                             # 列出存款金额
8  hungbank = banks.Shilin_Banks('Hung')               # 定义Shilin_Banks类对象
9  print("Hung's banks  = ", hungbank.bank_title())    # 打印银行名称
```

执行结果 与 ch13_8.py 相同。

13-4-5　模块内导入另一个模块的类

有时候可能一个模块内有太多类了，此时可以考虑将一系列的类分成两个或更多个模块存储。如果拆成类的模块之间彼此有衍生关系，则子类也需将父类导入，执行时才不会有错误产生。下面是将 Banks 模块拆成两个模块的内容。

程序实例 banks1.py：这个模块含父类 Banks 的内容。

```
1  # banks1.py
2  # 这是一个包含Banks类的模块(module)
3  class Banks():
4      # 定义银行类
5      def __init__(self, uname):                # 初始化方法
6          self.__name = uname                   # 设置私有存款者名字
7          self.__balance = 0                    # 设置私有开户金额是0
8          self.__title = "Taipei Bank"          # 设置私有银行名称
9
10     def save_money(self, money):              # 设计存款方法
11         self.__balance += money               # 执行存款
12         print("存款 ", money, " 完成")         # 打印存款完成
13
14     def withdraw_money(self, money):          # 设计取款方法
15         self.__balance -= money               # 执行取款
16         print("取款 ", money, " 完成")         # 打印取款完成
17
18     def get_balance(self):                    # 获得存款余额
19         print(self.__name.title(), " 目前余额: ", self.__balance)
20
21     def bank_title(self):                     # 获得银行名称
22         return self.__title
```

程序实例 shilin_banks.py：这个模块含子类 Shilin_Banks 的内容，读者应留意第 3 行，笔者在这个模块内导入了 banks1.py 模块的 Banks 类。

```
1  # shilin_banks.py
2  # 这是一个包含Shilin_Banks类的模块(module)
3  from banks1 import Banks                          # 导入Banks类
4
5  class Shilin_Banks(Banks):
6      # 定义士林分行
7      def __init__(self, uname):
8          self.title = "Taipei Bank - Shilin Branch"  # 定义分行名称
9      def bank_title(self):                         # 获得银行名称
10         return self.title
```

程序实例 ch13_13.py：这个程序中，在第 2 和 3 行分别导入两个模块，整个程序的执行内容与 ch13_8.py 相同。

```
1  # ch13_13.py
2  from banks1 import Banks                          # 导入banks模块的Banks类
3  from shilin_Banks import Shilin_Banks             # 导入Shilin_Banks模块的Shilin_Banks类
4
5  jamesbank = Banks('James')                        # 定义Banks类对象
6  print("James's banks = ", jamesbank.bank_title()) # 打印银行名称
7  jamesbank.save_money(500)                         # 存钱
8  jamesbank.get_balance()                           # 列出存款金额
9  hungbank = Shilin_Banks('Hung')                   # 定义Shilin_Banks类对象
10 print("Hung's banks  = ", hungbank.bank_title())  # 打印银行名称
```

执行结果 与 ch13_8.py 相同。

13-5 随机数 random 模块

随机数（Random number）是指平均散布在某区间的数字。随机数用途很广，最常见的应用是设计游戏时可以控制输出结果，其实赌场的老虎机就是靠它赚钱。本节将介绍 random 模块中最有用的 3 个方法，同时也会分析赌场赚钱的利器。

13-5-1 randint( )

这个方法可以随机产生指定区间的整数，它的语法如下。

```
randint(min, max)            # 可以产生 min(含)与 max(含)之间的整数值
```

程序实例 ch13_14.py：建立一个程序分别产生各 3 组在 1 ～ 100、500 ～ 1000、2000 ～ 3000 的数字。

```
1  # ch13_14.py
2  import random              # 导入模块random
3
4  n = 3
5  for i in range(n):
6      print("1-100     : ", random.randint(1, 100))
7
8  for i in range(n):
9      print("500-1000  : ", random.randint(500, 1000))
10
11 for i in range(n):
12     print("2000-3000 : ", random.randint(2000, 3000))
```

执行结果

```
===================== RESTART: D:\Python\ch13\ch13_14.py =====================
1-100      :  11
1-100      :  83
1-100      :  21
500-1000   :  619
500-1000   :  767
500-1000   :  976
2000-3000  :  2794
2000-3000  :  2043
2000-3000  :  2013
```

程序实例 ch13_15.py：猜数字游戏，这个程序首先会用 randint( ) 方法产生一个 1 ～ 10 的数字，然后如果猜的数值太小会要求猜大一些，然后如果猜的数值太大会要求猜小一些。

```
1  # ch13_15.py
2  import random                          # 导入模块random
3
4  min, max = 1, 10
5  ans = random.randint(min, max)         # 随机数产生答案
6  while True:
7      yourNum = int(input("请猜1-10之间数字: "))
8      if yourNum == ans:
9          print("恭喜!答对了")
10         break
11     elif yourNum < ans:
12         print("请猜大一些")
13     else:
14         print("请猜小一些")
```

执行结果

```
===================== RESTART: D:\Python\ch13\ch13_15.py =====================
请猜1-10之间数字: 5
请猜大一些
请猜1-10之间数字: 8
恭喜!答对了
```

一般赌场的机器可以用随机数控制输赢，例如，某个猜大小机器，一般人以为猜对率是 50%，但是只要控制随机数，赌场可以直接控制输赢比例。

程序实例 ch13_16.py：这是一个猜大小的游戏，程序执行初可以设置庄家的输赢比例，程序执行过程中会立即回应是否猜对。

```
1  # ch13_16.py
2  import random                                        # 导入模块random
3
4  min, max = 1, 100                                    # 随机数最小与最大值设定
5  winPercent = int(input("请输入庄家赢的比率(0-100)之间 :"))
6
7  while True:
8      print("猜大小游戏: L或l表示大,  S或s表示小, Q或q则程序结束")
9      customerNum = input("= ")                        # 读取玩家输入
10     if customerNum == 'Q' or customerNum == 'q':     # 若输入Q或q
11         break                                        # 程序结束
12     num = random.randint(min, max)                   # 产生是否让玩家答对的随机数
13     if num > winPercent:                             # 随机数在81~100间回应玩家猜对
14         print("恭喜!答对了\n")
15     else:                                            # 随机数在1~80间回应玩家猜错
16         print("答错了!请再试一次\n")
```

执行结果

```
==================== RESTART: D:\Python\ch13\ch13_16.py ====================
请输入庄家赢的比率(0~100)之间 :80
猜大小游戏: L或l表示大， S或s表示小，Q或q则程序结束
= l
恭喜!答对了

猜大小游戏: L或l表示大， S或s表示小，Q或q则程序结束
= s
答错了!请再试一次

猜大小游戏: L或l表示大， S或s表示小，Q或q则程序结束
= q
```

这个程序的关键点 1 是程序第 5 行，庄家可以在程序启动时先设置赢的比率。第 2 个关键点是程序第 12 行产生的随机数，由 1 ～ 100 的随机数决定玩家是赢或输，猜大小只是晃子。例如，庄家刚开始设置赢的概率是 80%，相当于如果随机数是在 81 ～ 100 算玩家赢，如果随机数是 1 ～ 80 算玩家输。



13-5-2 choice( )

这个方法可以在一个列表（list）中随机返回一个元素。

程序实例 ch13_17.py：有一个水果列表，使用 choice( ) 方法随机选取一个水果。

```
1  # ch13_17.py
2  import random                          # 导入模块random
3
4  fruits = ['苹果', '香蕉', '西瓜', '水蜜桃', '百香果']
5  print(random.choice(fruits))
```

执行结果 下列是程序执行两次的结果。

```
==================== RESTART: D:\Python\ch13\ch13_17.py ====================
水蜜桃
>>>
==================== RESTART: D:\Python\ch13\ch13_17.py ====================
西瓜
```

程序实例 ch13_17_1.py：骰子有 6 面，点数是 1 ～ 6，这个程序会产生 10 次 1 ～ 6 的值。

```
1  # ch13_17_1.py
2  import random                          # 导入模块random
3
4  for i in range(10):
5      print(random.choice([1,2,3,4,5,6]), end=",")
```

执行结果

```
==================== RESTART: D:/Python/ch13/ch13_17_1.py ====================
5,5,2,6,4,6,1,2,6,1,
```

13-5-3　shuffle( )

这个方法可以将列表元素重新排列，如果要设计扑克牌（Porker）游戏，在发牌前可以使用这个方法将牌打乱重新排列。

程序实例 ch13_18.py：将列表内的扑克牌次序打乱，然后重新排列。

```
1  # ch13_18.py
2  import random                        # 导入模块random
3
4  porker = ['2', '3', '4', '5', '6', '7', '8',
5            '9', '10', 'J', 'Q', 'K', 'A']
6  for i in range(3):
7      random.shuffle(porker)           # 将次序打乱重新排列
8      print(porker)
```

执行结果

```
==================== RESTART: D:\Python\ch13\ch13_18.py ====================
['7', '5', '10', '8', '2', 'A', '9', '3', 'Q', 'J', '4', 'K', '6']
['Q', '4', 'A', 'K', '10', '5', '6', '2', '3', '9', '7', '8', 'J']
['5', 'Q', '7', '8', '4', 'K', '2', '3', '9', '6', 'A', 'J', '10']
```

将列表元素打乱，很适合老师出防止作弊的考题。例如，如果有 50 位学生，为了避免学生偷窥邻座的考卷，可以将出好的题目处理成列表，然后使用 for 循环执行 50 次 shuffle( )，这样就可以得到 50 份考题相同但是次序不同的考卷。这个将当作读者的习题。

13-5-4　sample( )

sample( ) 的语法如下。

sample（列表，数量）

可以随机返回第 2 个参数数量的列表元素。

程序实例 ch13_18_1.py：设计大乐透彩票号码，大乐透号码是由 6 个 1 ～ 49 的数字组成，然后外加一个特别号，这个程序会产生 6 个号码以及一个特别号。

```
1  # ch13_18_1.py
2  import random                                # 导入模块random
3
4  lotterys = random.sample(range(1,50), 7)     # 7组号码
5  specialNum = lotterys.pop()                  # 特别号
6
7  print("第xxx期大乐透号码 ", end="")
8  for lottery in sorted(lotterys):             # 排序打印大乐透号码
9      print(lottery, end=" ")
10 print("\n特别号:%d" % specialNum)            # 打印特别号
```

执行结果

```
==================== RESTART: D:\Python\ch13\ch13_18_1.py ====================
第xxx期大乐透号码 1 20 28 31 35 39
特别号:29
```

13-5-5　uniform( )

uniform( ) 可以随机产生（x,y）之间的浮点数，它的语法格式如下。

uniform(x,y)

x 是随机数最小值，包含 x 值；y 是随机数最大值，不包含该值。

程序实例 ch13_18_2.py：产生 5 个 0 ～ 10 随机浮点数的应用。

```
1  # ch13_18_2.py
2  import random
3
4  for i in range(5):
5      print("uniform(1,10) : ", random.uniform(1, 10))
```

执行结果

```
==================== RESTART: D:/Python/ch13/ch13_18_2.py ====================
uniform(1,10) :  4.650312334612405
uniform(1,10) :  6.862453320095783
uniform(1,10) :  3.2055807663870484
uniform(1,10) :  2.712843194025017
uniform(1,10) :  7.5172219039912065
```

13-5-6　random( )

random( ) 可以随机产生 0.0（含）～ 1.0 的随机浮点数。

程序实例 ch13_18_3.py：产生 10 个 0.0 ～ 1.0 的随机浮点数。

```
1  # ch13_18_3.py
2  import random
3
4  for i in range(10):
5      print(random.random())
```

执行结果

```
==================== RESTART: D:/Python/ch13/ch13_18_3.py ====================
0.8265495242543065
0.7281007292756023
0.6852959134781751
0.9485994816643372
0.914983328764756
0.9846475437517717
0.9814128181047725
0.8033467190495047
0.13216803444569913
0.6610479743073929
```

13-6　时间 time 模块

13-6-1　time( )

time( ) 方法可以返回自 1970 年 1 月 1 日 00:00:00AM 以来的秒数，初看好像用处不大，其实如果想要掌握某段工作所花时间则是很有用的。例如，若应用于程序实例 ch13_15.py，可以用它计算

猜数字所花时间。

程序实例 ch13_19.py：计算自 1970 年 1 月 1 日 00:00:00AM 以来的秒数。

```
1  # ch13_19.py
2  import time                        # 导入模块time
3
4  print("计算1970年1月1日00:00:00至今的秒数 = ", int(time.time()))
```

执行结果

```
==================== RESTART: D:\Python\ch13\ch13_19.py ====================
计算1970年1月1日00:00:00至今的秒数 =  1559207734
```

读者的执行结果将和笔者不同，因为我们是在不同的时间点执行这个程序。

程序实例 ch13_20.py：扩充 ch13_15.py 的功能，主要是增加计算花多少时间猜对数字。

```
1  # ch13_20.py
2  import random                              # 导入模块random
3  import time                                # 导入模块time
4
5  min, max = 1, 10
6  ans = random.randint(min, max)             # 随机数产生答案
7  yourNum = int(input("请猜1~10之间数字: "))
8  starttime = int(time.time())               # 起始秒数
9  while True:
10     if yourNum == ans:
11         print("恭喜!答对了")
12         endtime = int(time.time())         # 结束秒数
13         print("所花时间: ", endtime - starttime, " 秒")
14         break
15     elif yourNum < ans:
16         print("请猜大一些")
17     else:
18         print("请猜小一些")
19     yourNum = int(input("请猜1~10之间数字: "))
```

执行结果

```
==================== RESTART: D:\Python\ch13\ch13_20.py ====================
请猜1~10之间数字: 5
请猜大一些
请猜1~10之间数字: 8
恭喜!答对了
所花时间:  2  秒
```

13-6-2 sleep( )

sleep( ) 方法可以让工作暂停，这个方法的参数单位是秒。这个方法对于设计动画非常有帮助，未来还会介绍这个方法更多的应用。

程序实例 ch13_21.py：每秒打印一次列表的内容。

```
1  # ch13_21.py
2  import time                        # 导入模块time
3
4  fruits = ['苹果', '香蕉', '西瓜', '水蜜桃', '百香果']
5  for fruit in fruits:
6      print(fruit)
7      time.sleep(1)                  # 暂停1秒
```

执行结果

```
===================== RESTART: D:\Python\ch13\ch13_21.py =====================
苹果
香蕉
西瓜
水蜜桃
百香果
```

13-6-3　asctime( )

这个方法会以可以阅读的方式列出目前的系统时间。

程序实例 ch13_22.py：列出目前系统时间。

```
1  # ch13_22.py
2  import time                              # 导入模块time
3
4  print(time.asctime())                    # 列出目前系统时间
```

执行结果

```
===================== RESTART: D:\Python\ch13\ch13_22.py =====================
Wed Nov 21 16:00:59 2018
```

13-6-4　localtime( )

这个方法可以返回目前时间的结构数据，所返回的结构可以用索引方式获得其中内容。

程序实例 ch13_23.py：使用 localtime( ) 方法列出目前时间的结构数据，同时使用索引列出其中内容。

```
1  # ch13_23.py
2  import time                         # 导入模块time
3
4  xtime = time.localtime()
5  print(xtime)                        # 列出目前系统时间
6  print("年 ", xtime[0])
7  print("月 ", xtime[1])
8  print("日 ", xtime[2])
9  print("时 ", xtime[3])
10 print("分 ", xtime[4])
11 print("秒 ", xtime[5])
12 print("星期几   ", xtime[6])
13 print("第几天   ", xtime[7])
14 print("夏令时间 ", xtime[8])
```

执行结果

```
===================== RESTART: D:\Python\ch13\ch13_23.py =====================
time.struct_time(tm_year=2019, tm_mon=5, tm_mday=30, tm_hour=17, tm_min=33, tm_s
ec=38, tm_wday=3, tm_yday=150, tm_isdst=0)
年  2019
月  5
日  30
时  17
分  33
秒  38
星期几    3
第几天    150
夏令时间  0
```

上述索引第 12 行中 [6] 代表星期几的设置，0 代表星期一，1 代表星期二，…上述第 13 行中索引 [7] 是第几天的设置，代表这是一年中的第几天。上述第 14 行中索引 [8] 是夏令时间的设置，0 代表不是，1 代表是。

13-7 系统 sys 模块

这个模块可以控制 Python Shell 窗口消息。

13-7-1 version 和 version_info 属性

这个属性可以列出目前所使用 Python 的版本消息。

程序实例 ch13_24.py：列出目前所使用 Python 的版本消息。

```
1  # ch13_24.py
2  import sys
3
4  print("目前Python版本是: ", sys.version)
5  print("目前Python版本是: ", sys.version_info)
```

执行结果

```
==================== RESTART: D:\Python\ch13\ch13_24.py ====================
目前Python版本是:  3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914
32 bit (Intel)]
目前Python版本是:  sys.version_info(major=3, minor=7, micro=0, releaselevel='fin
al', serial=0)
```

13-7-2 stdin 对象

stdin 是 standard input 的缩写，是指从屏幕输入（可想成 Python Shell 窗口），这个对象可以搭配 readline( ) 方法，读取屏幕输入直到按 Enter 键。

程序实例 ch13_25.py：读取屏幕输入。

```
1  # ch13_25.py
2  import sys
3  print("请输入字符串, 输入完按Enter = ", end = "")
4  msg = sys.stdin.readline()
5  print(msg)
```

执行结果

```
==================== RESTART: D:\Python\ch13\ch13_25.py ====================
请输入字符串, 输入完按Enter = Python王者归来
Python王者归来
```

在 readline( ) 方法内可以加上正整数参数，例如 readline(n)，这个 n 代表所读取的字符数，其中一个中文字或空格也算一个字符数。

程序实例 ch13_26.py：从屏幕读取 8 个字符数。

```
1  # ch13_26.py
2  import sys
3  print("请输入字符串, 输入完按Enter = ", end = "")
4  msg = sys.stdin.readline(8)           # 读8个字符
5  print(msg)
```

执行结果

```
===================== RESTART: D:\Python\ch13\ch13_26.py =====================
请输入字符串, 输入完按Enter = Python王者归来
Python王者
>>>
===================== RESTART: D:\Python\ch13\ch13_26.py =====================
请输入字符串, 输入完按Enter = I like Python
I like P
```

13-7-3　stdout 对象

stdout 是 standard ouput 的缩写，是指从屏幕输出（可想成 Python Shell 窗口），这个对象可以搭配 write( ) 方法，从屏幕输出数据。

程序实例 ch13_27.py：使用 stdout 对象输出数据。

```
1  # ch13_27.py
2  import sys
3
4  sys.stdout.write("I like Python")
```

执行结果

```
===================== RESTART: D:\Python\ch13\ch13_27.py =====================
I like Python
```

其实这个对象若是使用 Python Shell 窗口，最后会同时列出输出的字符数。

```
>>> import sys
>>> sys.stdout.write("I like Python")
I like Python13
>>>
```

13-7-4　platform 属性

可以返回目前 Python 的使用平台。

程序实例 ch13_27_1.py：列出笔者计算机的使用平台。

```
1  # ch13_27_1.py
2  import sys
3
4  print(sys.platform)
```

执行结果

```
==================== RESTART: D:\Python\ch13\ch13_27_1.py ====================
win32
```

13-7-5　path 属性

Python 的 sys.path 参数是一个列表数据，这个列表记录模块所在的目录，当使用 import 导入模块时，Python 会到此列表目录找寻文件，然后导入。

程序实例 ch13_27_2.py：列出笔者计算机目前环境变量 path 的值。

```
1  # ch13_27_2.py
2  import sys
3  for dirpath in sys.path:
4      print(dirpath)
```

执行结果

```
==================== RESTART: D:\Python\ch13\ch13_27_2.py ====================
D:\Python\ch13
C:\Users\User\AppData\Local\Programs\Python\Python37-32\Lib\idlelib
C:\Users\User\AppData\Local\Programs\Python\Python37-32\python37.zip
C:\Users\User\AppData\Local\Programs\Python\Python37-32\DLLs
C:\Users\User\AppData\Local\Programs\Python\Python37-32\lib
C:\Users\User\AppData\Local\Programs\Python\Python37-32
C:\Users\User\AppData\Local\Programs\Python\Python37-32\lib\site-packages
```

读者可以看到笔者计算机所列出 sys.path 的内容，当导入模块时 Python 会依上述顺序往下查找所导入的模块，当找到第一个时就会导入。上述 sys.path 第 0 个元素是 D:\Python\ch13，这是笔者所设计模块的目录，如果笔者不小心设计了相同的系统模块，例如 time，同时它的查找路径在标准 Python 链接库的模块路径前面，将造成程序无法存取标准链接库的模块。

13-7-6　getwindowsversion( )

返回目前 Python 安装环境的 Windows 操作系统版本。

程序实例 ch13_27_3.py：列出目前的 Windows 操作系统版本。

```
1  # ch13_27_3.py
2  import sys
3
4  print(sys.getwindowsversion())
```

执行结果

```
==================== RESTART: D:\Python\ch13\ch13_27_3.py ====================
sys.getwindowsversion(major=10, minor=0, build=17134, platform=2, service_pack='
')
```

13-7-7　executable

列出目前所使用 Python 的可执行文件路径。

程序实例 ch13_27_4.py：列出笔者计算机 Python 的可执行文件路径。

```
1  # ch13_27_4.py
2  import sys
3
4  print(sys.executable)
```

执行结果

```
==================== RESTART: D:/Python/ch13/ch13_27_4.py ====================
C:\Users\cshun\AppData\Local\Programs\Python\Python37-32\pythonw.exe
```

13-7-8　获得 getrecursionlimit( ) 与设置 setrecursionlimit( ) 循环次数

在 11-7 节已经说明 sys.setrecursionlimit( ) 可以获得目前 Python 的循环次数，sys.setcursionlimit(x) 则是可以设置目前 Python 的循环次数，参数 x 是循环次数。

```
>>> import sys
>>> sys.setrecursionlimit(100)
>>> sys.getrecursionlimit()
100
```

13-7-9　DOS 命令行自变量

有时候设计一些程序必须在 DOS 命令行执行，命令行上所输入的自变量会以列表形式记录在 sys.argv 内。

程序实例 ch13_27_5.py：列出命令行自变量。

```
1  # ch13_27_5.py
2  import sys
3  print("命令行参数 : ", sys.argv)
```

执行结果

```
PS C:\Users\User> python d:\Python\ch13\ch13_27_5.py
命令行参数 :  ['d:\\Python\\ch13\\ch13_27_5.py']
PS C:\Users\User> python d:\Python\ch13\ch13_27_5.py Hello! Hi Good-bye
命令行参数 :  ['d:\\Python\\ch13\\ch13_27_5.py', 'Hello!', 'Hi', 'Good-bye']
```

13-8　keyword 模块

这个模块有一些 Python 关键词的功能。

13-8-1　kwlist 属性

这个属性包含所有 Python 的关键词。

程序实例 ch13_28.py：列出所有 Python 关键词。

```
1  # ch13_28.py
2  import keyword
3
4  print(keyword.kwlist)
```

执行结果

```
==================== RESTART: D:\Python\ch13\ch13_28.py ====================
['False', 'None', 'True', 'and', 'as', 'assert', 'async', 'await', 'break', 'cla
ss', 'continue', 'def', 'del', 'elif', 'else', 'except', 'finally', 'for', 'from
', 'global', 'if', 'import', 'in', 'is', 'lambda', 'nonlocal', 'not', 'or', 'pas
s', 'raise', 'return', 'try', 'while', 'with', 'yield']
```

13-8-2 iskeyword( )

这个方法可以返回参数的字符串是否是关键词，如果是返回 True，如果否返回 False。

程序实例 ch13_29.py：检查列表内的字是否是关键词。

```
1  # ch13_29.py
2  import keyword
3
4  keywordLists = ['as', 'while', 'break', 'sse', 'Python']
5  for x in keywordLists:
6      print("%8s " % x, keyword.iskeyword(x))
```

执行结果

```
==================== RESTART: D:/Python/ch13/ch13_29.py ====================
      as  True
   while  True
   break  True
     sse  False
  Python  False
```

13-9 日期 calendar 模块

日期模块有一些日历数据，很方便使用，本节将介绍几个常用的方法，使用此模块前需要先 import calendar。

13-9-1 列出某年是否闰年 isleap( )

如果是闰年返回 True，否则返回 False。

程序实例 ch13_30.py：分别列出 2020 年和 2021 年是否闰年。

```
1  # ch13_30.py
2  import calendar
3
4  print("2020年是否闰年", calendar.isleap(2020))
5  print("2021年是否闰年", calendar.isleap(2021))
```

执行结果

```
==================== RESTART: D:\Python\ch13\ch13_30.py ====================
2020年是否闰年 True
2021年是否闰年 False
```

13-9-2　打印月历 month( )

这个方法完整的参数是 month(year,month)，可以列出指定年份月份的月历。

程序实例 ch13_31.py：列出 2020 年 1 月的月历。

```
1  # ch13_31.py
2  import calendar
3
4  print(calendar.month(2020,1))
```

执行结果

```
==================== RESTART: D:/Python/ch13/ch13_31.py ====================
    January 2020
Mo Tu We Th Fr Sa Su
       1  2  3  4  5
 6  7  8  9 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30 31
```

13-9-3　打印年历 calendar( )

这个方法完整的参数是 calendar(year)，可以列出指定年份的年历。

程序实例 ch13_32.py：列出 2020 年的年历。

```
1  # ch13_32.py
2  import calendar
3
4  print(calendar.calendar(2020))
```

执行结果

```
==================== RESTART: D:/Python/ch13/ch13_32.py ====================
                                  2020

      January                   February                   March
Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su
       1  2  3  4  5                      1  2                         1
 6  7  8  9 10 11 12       3  4  5  6  7  8  9       2  3  4  5  6  7  8
13 14 15 16 17 18 19      10 11 12 13 14 15 16       9 10 11 12 13 14 15
20 21 22 23 24 25 26      17 18 19 20 21 22 23      16 17 18 19 20 21 22
27 28 29 30 31            24 25 26 27 28 29         23 24 25 26 27 28 29
                                                    30 31

       April                      May                       June
Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su
       1  2  3  4  5                   1  2  3       1  2  3  4  5  6  7
 6  7  8  9 10 11 12       4  5  6  7  8  9 10       8  9 10 11 12 13 14
13 14 15 16 17 18 19      11 12 13 14 15 16 17      15 16 17 18 19 20 21
20 21 22 23 24 25 26      18 19 20 21 22 23 24      22 23 24 25 26 27 28
27 28 29 30               25 26 27 28 29 30 31      29 30

        July                     August                  September
Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su
       1  2  3  4  5                      1  2          1  2  3  4  5  6
 6  7  8  9 10 11 12       3  4  5  6  7  8  9       7  8  9 10 11 12 13
13 14 15 16 17 18 19      10 11 12 13 14 15 16      14 15 16 17 18 19 20
20 21 22 23 24 25 26      17 18 19 20 21 22 23      21 22 23 24 25 26 27
27 28 29 30 31            24 25 26 27 28 29 30      28 29 30
                          31

      October                   November                  December
Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su
          1  2  3  4                         1          1  2  3  4  5  6
 5  6  7  8  9 10 11       2  3  4  5  6  7  8       7  8  9 10 11 12 13
12 13 14 15 16 17 18       9 10 11 12 13 14 15      14 15 16 17 18 19 20
19 20 21 22 23 24 25      16 17 18 19 20 21 22      21 22 23 24 25 26 27
26 27 28 29 30 31         23 24 25 26 27 28 29      28 29 30 31
                          30
```

13-10 几个增强 Python 功力的模块

13-10-1 collections 模块

1. defaultdict( )

这个方法可以为新建立的字典设置默认值，它的参数是一个函数，如果参数是 int，则参数相当于是 int( )，默认值会返回 0。如果参数是 list 或 dict，默认值是分别返回“[ ]”或“{ }”。如果省略参数，会返回 None。

程序实例 ch13_33.py：使用 defaultdict( ) 建立字典的应用。

```
1  # ch13_33.py
2  from collections import defaultdict
3  fruits = defaultdict(int)
4  fruits["apple"] = 20
5  fruits["orange"]           # 使用int预设的0
6  print(fruits["apple"])
7  print(fruits["orange"])
8  print(fruits)
```

执行结果

```
==================== RESTART: D:/Python/ch13/ch13_33.py ====================
20
0
defaultdict(<class 'int'>, {'apple': 20, 'orange': 0})
```

除了使用 int、list … 外，也可以自行设计 defaultdict( ) 方法内的函数。

程序实例 ch13_34.py：使用自行设计的函数重新设计程序实例 ch13_33.py。

```
1  # ch13_34.py
2  from collections import defaultdict
3  def price():
4      return 10
5
6  fruits = defaultdict(price)
7  fruits["apple"] = 20
8  fruits["orange"]           # 使用自行设计的price()
9  print(fruits["apple"])
10 print(fruits["orange"])
11 print(fruits)
```

执行结果

```
==================== RESTART: D:/Python/ch13/ch13_34.py ====================
20
10
defaultdict(<function price at 0x02F20420>, {'apple': 20, 'orange': 10})
```

程序实例 ch13_35.py：使用 lambda 重新设计 ch13_34.py。

```
1  # ch13_35.py
2  from collections import defaultdict
3
4  fruits = defaultdict(lambda:10)
5  fruits["apple"] = 20
6  fruits["orange"]              # 使用lambda设置的10
7  print(fruits["apple"])
8  print(fruits["orange"])
9  print(fruits)
```

执行结果 与 ch13_34.py 相同。

当使用 defaultdict(int) 时，也就是参数放 int 时，可以利用此特性建立计数器。

程序实例 ch13_36.py：利用参数是 int 的特性建立计数器。

```
1  # ch13_36.py
2  from collections import defaultdict
3
4  fruits = defaultdict(int)
5  for fruit in ["apple","orange","apple"]:
6      fruits[fruit] += 1
7
8  for fruit, count in fruits.items():
9      print(fruit, count)
```

执行结果

```
==================== RESTART: D:/Python/ch13/ch13_36.py ====================
apple 2
orange 1
```

对于 ch13_36.py 而言，如果改成第 9 章的 dict，使用上述第 6 行的写法会有 KeyError 错误，因为尚未建立该键，必须使用下列方式改写。

程序实例 ch13_37.py：使用传统 dict 字典方式重新设计 ch13_36.py。

```
 1  # ch13_37.py
 2
 3  fruits = {}
 4  for fruit in ["apple","orange","apple"]:
 5      if not fruit in fruits:
 6          fruits[fruit] = 0
 7      fruits[fruit] += 1
 8
 9  for fruit, count in fruits.items():
10      print(fruit, count)
```

执行结果 与 ch13_36.py 相同。

2. Counter( )

这个方法可以将列表元素转成字典的键，字典的值则是元素在列表中出现的次数。留意：此方法所建的数据类型是 Collections.Counter，元素则是字典。

程序实例 ch13_38.py：使用 Counter( ) 将列表转成字典的应用。

```
1  # ch13_38.py
2  from collections import Counter
3
4  fruits = ["apple","orange","apple"]
5  fruitsdict = Counter(fruits)
6  print(fruitsdict)
```

执行结果

```
==================== RESTART: D:/Python/ch13/ch13_38.py ====================
Counter({'apple': 2, 'orange': 1})
```

3. most_common( )

这个 most_common(n) 方法如果省略参数 n，可以参考键 : 值的数量由大排到小返回。n 是设置返回多少个元素。

程序实例 ch13_39.py：most_common( ) 的应用。

```
1  # ch13_39.py
2  from collections import Counter
3
4  fruits = ["apple","orange","apple"]
5  fruitsdict = Counter(fruits)
6  myfruits1 = fruitsdict.most_common()
7  print(myfruits1)
8  myfruits0 = fruitsdict.most_common(0)
9  print(myfruits0)
10 myfruits1 = fruitsdict.most_common(1)
11 print(myfruits1)
12 myfruits2 = fruitsdict.most_common(2)
13 print(myfruits2)
```

执行结果

```
==================== RESTART: D:/Python/ch13/ch13_39.py ====================
[('apple', 2), ('orange', 1)]
[]
[('apple', 2)]
[('apple', 2), ('orange', 1)]
```

4. Counter 对象的加与减

对于 Counter 对象而言，可以使用加法 + 与减法 -。将两个对象相加，相加的方式是所有元素相加，若是有重复的元素则键的值会相加。或是如果想列出 A 有但 B 没有的元素，可以使用 A – B。

程序实例 ch13_40.py：执行 Counter 对象相加，同时将 fruitsdictA 有的但是 fruitsdictB 没有的列出来。

```
1  # ch13_40.py
2  from collections import Counter
3
4  fruits1 = ["apple","orange","apple"]
5  fruitsdictA = Counter(fruits1)
6  fruits2 = ["grape","orange","orange", "grape"]
7  fruitsdictB = Counter(fruits2)
8  # 加法
9  fruitsdictAdd = fruitsdictA + fruitsdictB
10 print(fruitsdictAdd)
11 # 减法
12 fruitsdictSub = fruitsdictA - fruitsdictB
13 print(fruitsdictSub)
```

执行结果

```
===================== RESTART: D:/Python/ch13/ch13_40.py =====================
Counter({'orange': 3, 'apple': 2, 'grape': 2})
Counter({'apple': 2})
```

5. Counter 对象的交集与联集

可以使用 & 当作交集符号，| 是联集符号。联集与加法不一样，它不会将数量相加，只是取多的部分。交集则是取数量少的部分。

程序实例 ch13_41.py：交集与联集的应用。

```
1  # ch13_41.py
2  from collections import Counter
3
4  fruits1 = ["apple","orange","apple"]
5  fruitsdictA = Counter(fruits1)
6  fruits2 = ["grape","orange","orange", "grape"]
7  fruitsdictB = Counter(fruits2)
8  # 交集
9  fruitsdictInter = fruitsdictA & fruitsdictB
10 print(fruitsdictInter)
11 # 联集
12 fruitsdictUnion = fruitsdictA | fruitsdictB
13 print(fruitsdictUnion)
```

执行结果

```
===================== RESTART: D:/Python/ch13/ch13_41.py =====================
Counter({'orange': 1})
Counter({'apple': 2, 'orange': 2, 'grape': 2})
```

6. deque( )

这是数据结构中的双头序列，具有堆栈 stack 与序列 queue 的功能，可以从左右两边增加元素，也可以从左右两边删除元素。pop( ) 方法可以移除右边的元素并返回，popleft( ) 可以移除左边的元素并返回。

程序实例 ch13_42.py：在程序设计中有一个常用的名词“回文（palindrome）”。对于一个字符串，从左右两边往内移动，如果相同就一直比对到中央，如果全部相同就是回文，否则不是回文。

```
1  # ch13_42.py
2  from collections import deque
3
4  def palindrome(word):
5      wd = deque(word)
6      while len(wd) > 1:
7          if wd.pop() != wd.popleft():
8              return False
9      return True
10
11 print(palindrome("x"))
12 print(palindrome("abccba"))
13 print(palindrome("radar"))
14 print(palindrome("python"))
```

执行结果

```
===================== RESTART: D:/Python/ch13/ch13_42.py =====================
True
True
True
False
```

另一种简单的方式是使用 [::-1]，可以将字符串反转，直接比较就可以判断是否回文。

程序实例 ch13_43.py：使用字符串反转判断是否回文。

```
1  # ch13_43.py
2  from collections import deque
3
4  def palindrome(word):
5      return word == word[::-1]
6
7  print(palindrome("x"))
8  print(palindrome("abccba"))
9  print(palindrome("radar"))
10 print(palindrome("python"))
```

执行结果 与 ch13_42.py 相同。

13-10-2 pprint 模块

之前所有程序都是使用 print( ) 做输出，输出原则是在 Python Shell 中输出，一行满了才跳到下一行输出。pprint( ) 的用法与 print( ) 相同，不过 pprint( ) 会执行一行输出一个元素，结果比较容易阅读。

程序实例 ch13_44.py：程序 ch13_27_2.py 输出 sys.path 的数据，当时为了执行结果看起来比较清爽，笔者使用 for 循环方式一次输出一个数据，其实使用 pprint( ) 可以获得几乎同样的结果。下面是比较 print( ) 与 pprint( ) 的结果。

```
1  # ch13_44.py
2  import sys
3  from pprint import pprint
4  print("使用print")
5  print(sys.path)
6  print("使用pprint")
7  pprint(sys.path)
```

执行结果

```
===================== RESTART: D:/Python/ch13/ch13_44.py =====================
使用print
['D:/Python/ch13', 'C:\\Users\\User\\AppData\\Local\\Programs\\Python\\Python37-
32\\Lib\\idlelib', 'C:\\Users\\User\\AppData\\Local\\Programs\\Python\\Python37-
32\\python37.zip', 'C:\\Users\\User\\AppData\\Local\\Programs\\Python\\Python37-
32\\DLLs', 'C:\\Users\\User\\AppData\\Local\\Programs\\Python\\Python37-32\\lib'
, 'C:\\Users\\User\\AppData\\Local\\Programs\\Python\\Python37-32', 'C:\\Users\\
User\\AppData\\Local\\Programs\\Python\\Python37-32\\lib\\site-packages']
使用pprint
['D:/Python/ch13',
 'C:\\Users\\User\\AppData\\Local\\Programs\\Python\\Python37-32\\Lib\\idlelib',
 'C:\\Users\\User\\AppData\\Local\\Programs\\Python\\Python37-32\\python37.zip',
 'C:\\Users\\User\\AppData\\Local\\Programs\\Python\\Python37-32\\DLLs',
 'C:\\Users\\User\\AppData\\Local\\Programs\\Python\\Python37-32\\lib',
 'C:\\Users\\User\\AppData\\Local\\Programs\\Python\\Python37-32',
 'C:\\Users\\User\\AppData\\Local\\Programs\\Python\\Python37-32\\lib\\site-pack
ages']
```

13-10-3　itertools 模块

1. chain( )

这个方法可以将 chain( ) 参数的元素内容一一迭代出来。

程序实例 ch13_45.py：chain( ) 的应用。

```
1  # ch13_45.py
2  import itertools
3  for i in itertools.chain([1,2,3],('a','d')):
4      print(i)
```

执行结果

```
==================== RESTART: D:/Python/ch13/ch13_45.py ====================
1
2
3
a
d
```

2. cycle( )

这个方法会产生无限迭代。

程序实例 ch13_46.py：cycle( ) 的应用。

```
1  # ch13_46.py
2  import itertools
3  for i in itertools.cycle(('a','b','c')):
4      print(i)
```

执行结果　可以按 Ctrl+C 组合键让程序中断。

```
==================== RESTART: D:/Python/ch13/ch13_46.py ====================
a
b
c
a
b
```

3. accumulate( )

如果 accumulate( ) 只有一个参数，则是列出累计的值。如果 accumulate( ) 有两个参数，则第 2 个参数是函数，可以用此函数列出累计的计算结果。

程序实例 ch13_47.py：accumulate( ) 的应用。

```
1  # ch13_47.py
2  import itertools
3  def mul(x, y):
4      return (x * y)
5  for i in itertools.accumulate((1,2,3,4,5)):
6      print(i)
7
8  for i in itertools.accumulate((1,2,3,4,5),mul):
9      print(i)
```

执行结果

```
===================== RESTART: D:/Python/ch13/ch13_47.py =====================
1
3
6
10
15
1
2
6
24
120
```

13-11 专题——赌场游戏骗局 / 蒙特卡罗模拟 / 文件加密

13-11-1 赌场游戏骗局

全球每一家赌场都装潢得很漂亮，各种噱头让我们想一窥内部。其实绝大部分的赌场有关计算机控制的机台都是可以作弊的，读者可以想想如果是依照 1∶1 的比例输赢，赌场哪来的费用支付员工薪资、美丽的装潢。ch13_7.py 设计了赌大小的游戏，程序开始即可以设置庄家的输赢比例，在这种状况下玩家以为自己手气背，其实非也，只是机台已被控制。

程序实例 ch13_48.py：这是 ch13_16.py 的扩充，刚开始玩家有 300 美元赌本，每次赌注是 100 美元，如果猜对赌金增加 100 美元，如果猜错赌金减少 100 美元，赌金没了，或是输入 Q 或 q 则程序结束。

```
1  # ch13_48.py
2  import random                                    # 导入模块random
3  money = 300                                      # 赌金总额
4  bet = 100                                        # 赌注
5  min, max = 1, 100                                # 随机数最小与最大值设定
6  winPercent = int(input("请输入庄家赢的比率(0-100)之间 :"))
7
8  while True:
9      print("欢迎光临 : 目前筹码金额 %d 美元 " % money)
10     print("每次赌注 %d 美元 " % bet)
11     print("猜大小游戏: L或l表示大，  S或s表示小，Q或q则程序结束")
12     customerNum = input("= ")                    # 读取玩家输入
13     if customerNum == 'Q' or customerNum == 'q': # 若输入Q或q
14         break                                    # 程序结束
15     num = random.randint(min, max)               # 产生是否让玩家答对的随机数
16     if num > winPercent:                         # 随机数在此区间回应玩家猜对
17         print("恭喜!答对了\n")
18         money += bet                             # 赌金总额增加
19     else:                                        # 随机数在此区间回应玩家猜错
20         print("答错了!请再试一次\n")
21         money -= bet                             # 赌金总额减少
22     if money <= 0:
23         break
24
25 print("欢迎下次再来")
```

执行结果

```
==================== RESTART: D:\Python\ch13\ch13_48.py ====================
请输入庄家赢的比率(0-100)之间 :90
欢迎光临 : 目前筹码金额 300 美元
每次赌注 100 美元
猜大小游戏: L或l表示大,  S或s表示小, Q或q则程序结束
= l
答错了!请再试一次

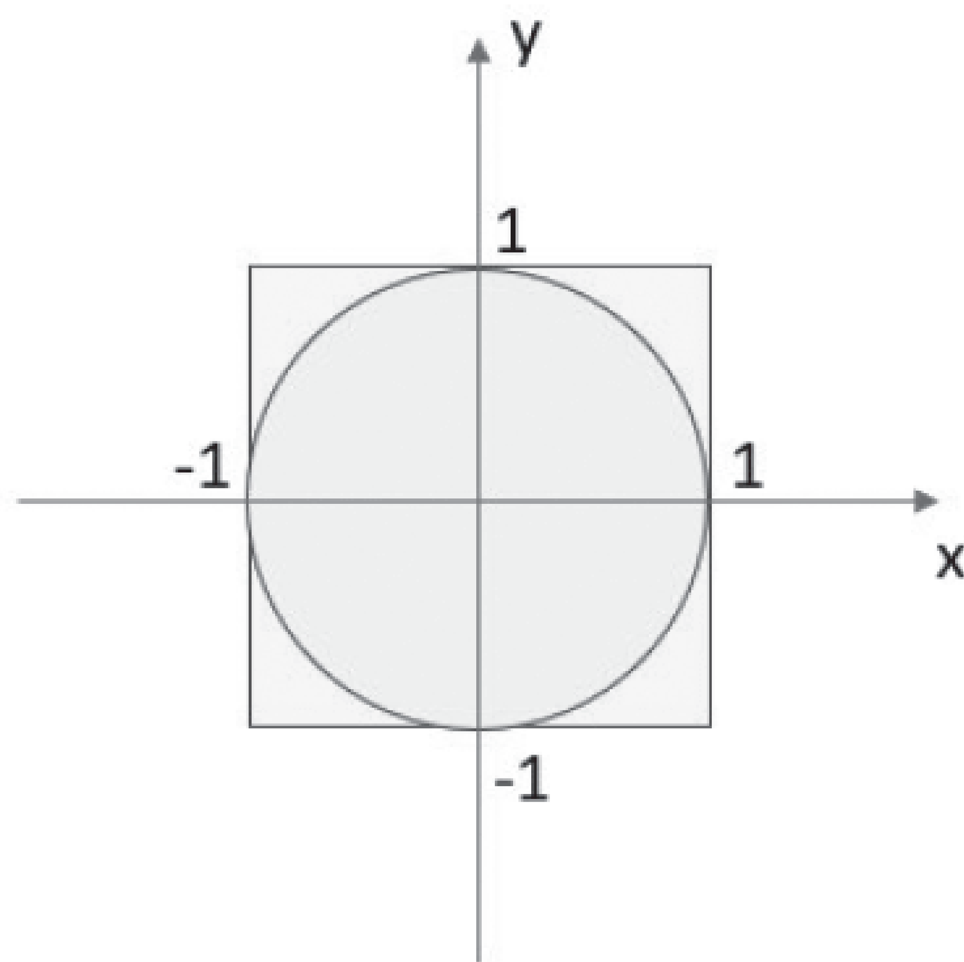
欢迎光临 : 目前筹码金额 200 美元
每次赌注 100 美元
猜大小游戏: L或l表示大,  S或s表示小, Q或q则程序结束
= l
答错了!请再试一次

欢迎光临 : 目前筹码金额 100 美元
每次赌注 100 美元
猜大小游戏: L或l表示大,  S或s表示小, Q或q则程序结束
= s
答错了!请再试一次

欢迎下次再来
```

13-11-2　蒙特卡罗模拟

可以使用蒙特卡罗模拟计算 PI 值，首先绘制一个外接正方形的圆，圆的半径是 1。



由上图可以知道，矩形面积是 4，圆面积是 PI。

如果现在要产生 1000000 个落在方形内的点，可以由下列公式计算点落在圆内的概率。

圆面积 / 矩形面积 = PI / 4

落在圆内的点个数 (Hits) = 1000000×PI / 4

如果落在圆内的点个数用 Hits 代替，则可以使用下列方式计算 PI。

PI = 4×Hits / 1000000

程序实例 ch13_49.py：蒙特卡罗模拟随机数计算 PI 值，这个程序会产生 100 万个随机点。

```
1  # ch13_49.py
2  import random
3
4  trials = 1000000
5  Hits = 0
6  for i in range(trials):
7      x = random.random() * 2 - 1     # x轴坐标
8      y = random.random() * 2 - 1     # y轴坐标
9      if x * x + y * y <= 1:          # 判断是否在圆内
10         Hits += 1
11 PI = 4 * Hits / trials
12
13 print("PI = ", PI)
```

执行结果

```
===================== RESTART: D:\Python\ch13\ch13_49.py =====================
PI =  3.143156
```

13-11-3 再谈文件加密

在 9-8-4 节已经讲解过文件加密，有一个模块 string，这个模块有一个属性是 printable，这个属性可以列出所有 ASCII 的可打印字符。

```
>>> import string
>>> string.printable
'0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ!"#$%&\'()*+,-./:
;<=>?@[\\]^_`{|}~ \t\n\r\x0b\x0c'
```

上述字符串最大的优点是可以处理所有的文件内容，所以在加密编码时已经可以应用于所有文件。在上述字符中最后几个是转义字符，可以参考 3-4-3 节，在做编码加密时可以将这些字符排除。

```
>>> abc = string.printable[:-5]
>>> abc
'0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ!"#$%&\'()*+,-./:
;<=>?@[\\]^_`{|}~ '
```

程序实例 ch13_50.py：设计一个加密函数，然后为字符串执行加密，所加密的字符串在第 16 行设置，取材自 1-11 节 Python 之禅的内容。

```
 1  # ch13_50.py
 2  import string
 3
 4  def encrypt(text, encryDict):              # 加密文件
 5      cipher = []
 6      for i in text:                         # 执行每个字符加密
 7          v = encryDict[i]                   # 加密
 8          cipher.append(v)                   # 加密结果
 9      return ''.join(cipher)                 # 将串行转成字符串
10
11  abc = string.printable[:-5]                # 取消不可打印字符
12  subText = abc[-3:] + abc[:-3]              # 加密字符串
13  encry_dict = dict(zip(subText, abc))       # 建立字典
14  print("打印编码字典\n", encry_dict)        # 打印字典
15
16  msg = 'If the implementation is easy to explain, it may be a good idea.'
17  ciphertext = encrypt(msg, encry_dict)
18
19  print("原始字符串 ", msg)
20  print("加密字符串 ", ciphertext)
```

执行结果

```
===================== RESTART: D:\Python\ch13\ch13_50.py =====================
打印编码字典
 {'}': '0', '~': '1', ' ': '2', '0': '3', '1': '4', '2': '5', '3': '6', '4': '7
', '5': '8', '6': '9', '7': 'a', '8': 'b', '9': 'c', 'a': 'd', 'b': 'e', 'c': 'f
', 'd': 'g', 'e': 'h', 'f': 'i', 'g': 'j', 'h': 'k', 'i': 'l', 'j': 'm', 'k': 'n
', 'l': 'o', 'm': 'p', 'n': 'q', 'o': 'r', 'p': 's', 'q': 't', 'r': 'u', 's': 'v
', 't': 'w', 'u': 'x', 'v': 'y', 'w': 'z', 'x': 'A', 'y': 'B', 'z': 'C', 'A': 'D
', 'B': 'E', 'C': 'F', 'D': 'G', 'E': 'H', 'F': 'I', 'G': 'J', 'H': 'K', 'I': 'L
', 'J': 'M', 'K': 'N', 'L': 'O', 'M': 'P', 'N': 'Q', 'O': 'R', 'P': 'S', 'Q': 'T
', 'R': 'U', 'S': 'V', 'T': 'W', 'U': 'X', 'V': 'Y', 'W': 'Z', 'X': '!', 'Y': '"
', 'Z': '#', '!': '$', '"': '%', '#': '&', '$': "'", '%': '(', '&': ')', "'": '*
', '(': '+', ')': ',', '*': '-', '+': '.', ',': '/', '-': ':', '.': ';', '/': '<
', ':': '=', ';': '>', '<': '?', '=': '@', '>': '[', '?': '\\', '@': ']', '[': '^
', '\\': '_', ']': '`', '^': '{', '_': '|', '`': '}', '{': '~', '|': ' '}
原始字符串  If the implementation is easy to explain, it may be a good idea.
加密字符串  Li2wkh2lpsohphqwdwlrq2lv2hdvB2wr2hAsodlq/2lw2pdB2eh2d2jrrg2lghd;
```

可以加密就可以解密，解密的字典基本上是将加密字典的键与值互换即可，如下所示。至于完整的程序设计将是读者的习题。

```
decry_dict = dict(zip(abc, subText))
```

13-11-4　只有自己可以破解的加密程序

上述加密字符间有一定规律，所以若是碰上高手可以破解此加密规则。如果想设计一个只有自己可以破解的加密程序，在程序实例 ch13_50.py 第 12 行可以使用下列方式处理。

```
newAbc = abc[:]                          # 产生新字符串复制
abllist = list(newAbc)                   # 字符串转成列表
random.shuffle(abclist)                  # 重排列表内容
subText = ''.join(abclist)               # 列表转成字符串
```

上述语句相当于打乱字符的对应顺序，如果这样做就必须将上述 subText 存储至数据库内，也就是保存字符打乱的顺序，否则连你自己未来也无法破解此加密结果。

程序实例 ch13_51.py：无法破解的加密程序，这个程序每次执行都会有不同的加密效果。

```
1  # ch13_51.py
2  import string
3  import random
4  def encrypt(text, encryDict):               # 加密文件
5      cipher = []
6      for i in text:                          # 执行每个字符加密
7          v = encryDict[i]                    # 加密
8          cipher.append(v)                    # 加密结果
9      return ''.join(cipher)                  # 将列表转成字符串
10
11 abc = string.printable[:-5]                 # 取消不可打印字符
12 newAbc = abc[:]                             # 产生新字符串复制
13 abclist = list(newAbc)                      # 转成列表
14 random.shuffle(abclist)                     # 打乱列表顺序
15 subText = ''.join(abclist)                  # 转成字符串
16 encry_dict = dict(zip(subText, abc))        # 建立字典
17 print("打印编码字典\n", encry_dict)          # 打印字典
18
19 msg = 'If the implementation is easy to explain, it may be a good idea.'
20 ciphertext = encrypt(msg, encry_dict)
21
22 print("原始字符串 ", msg)
23 print("加密字符串 ", ciphertext)
```

执行结果　下面是两次执行后显示不同的结果。

```
===================== RESTART: D:\Python\ch13\ch13_51.py =====================
打印编码字典
 {'>': '0', '^': '1', 'Q': '2', 'Z': '3', '"': '4', '|': '5', 'L': '6', 'b': '7'
, 'm': '8', 'K': '9', 'U': 'a', '3': 'b', 'C': 'c', 'p': 'd', 'c': 'e', 'j': 'f'
, 'f': 'g', 'q': 'h', ';': 'i', ' ': 'j', '*': 'k', 'B': 'l', '=': 'm', ']': 'n'
, 'y': 'o', 'N': 'p', 'x': 'q', 'H': 'r', 's': 's', 'V': 't', 'd': 'u', 'w': 'v'
, '@': 'w', 'k': 'x', 'A': 'y', '~': 'z', 'T': 'A', '$': 'B', 'v': 'C', 'F': 'D'
, '8': 'E', ',': 'F', 'M': 'G', '\\': 'H', 'J': 'I', '6': 'J', "'": 'K', 'u': 'L
', 'a': 'M', '<': 'N', '2': 'O', 'W': 'P', 'O': 'Q', '(': 'R', 'g': 'S', 'e': 'T
', '.': 'U', '&': 'V', '-': 'W', '0': 'X', 'P': 'Y', 'G': 'Z', '4': '!', '%': '"
', 't': '#', 'R': '$', '_': '%', ':': '&', 'i': "'", '?': '(', 'I': ')', 'h': '*
', 'S': '+', ')': ',', 'X': '-', '+': '.', '9': '/', '5': ':', '{': ';', '!': '<
', '#': '=', 'r': '>', '[': '?', ']': '@', 'D': '[', '1': '\\', 'n': ']', '/': '
^', 'o': '_', 'z': '`', '7': '{', '}': '|', 'Y': '}', 'E': '~', '`': ' '}
原始字符串  If the implementation is easy to explain, it may be a good idea.
加密字符串  )gj#*Tj'8dnT8T]#M#'_]j'sjTMsoj#_jTqdnM']Fj'#j8Moj7TjMjS__uj'uTMU
```

```
原始字符串  If the implementation is easy to explain, it may be a good idea.
加密字符串  ]cb0nc3Y(1nYngbeb38gc3AcneAfcb8cnh(1e3g6c3bcYefcdncec~88yc3yneD
```

由上述执行结果可以发现，加密结果更乱、更难理解，如何验证上述加密是正确的，这将是读者的习题。

习题

1. 请扩充 makefood 模块，增加 make_noodle( ) 函数，这个函数的第一个参数是面的种类，例如，牛肉面、肉丝面等，第二到多个参数则是自选配料，然后参考 ch13_2.py 调用方式，产生结果。(13-2 节)

```
===================== RESTART: D:\Python\ex\ex13_1.py =====================
牛肉面 的配料如下:
---  酸菜
---  辣酱
---  葱花
肉丝面 的配料如下:
---  辣酱
---  葱花
```

2. 请建立一个模块，这个模块含 4 个运算的类，分别是加法、减法、乘法和除法，运算完成后需返回结果。基本上每个方法都是含两个参数，运算原则是：

参数 1　op　参数 2

请分别用两组数字测试这个模块。(13-4 节)

```
===================== RESTART: D:\Python\ex\ex13_2.py =====================
请输入运算
1:加法
2:减法
3:乘法
4:除法
输入1/2/3/4: 1
a = 10
b = 5
a + b =  15
```

3. 请重新设计 ch13_15.py，将所猜数值改为 0 ～ 30，增加猜几次才答对，若是输入 Q 或 q，程序可直接结束。(13-5 节)

```
===================== RESTART: D:\Python\ex\ex13_3.py =====================
请猜1~30之间数字: 15
请猜大一些
请猜1~30之间数字: 23
请猜大一些
请猜1~30之间数字: 27
请猜大一些
请猜1~30之间数字: 29
恭喜!答对了
总共猜测 4 次
```

4. 在赌场有掷骰子机器，每次有 3 个骰子，可以压大或小、总计数字或是针对猜对数字获得理赔，请设计一个程序可以每次获得 3 组数字，然后列出结果。(13-5 节)

```
===================== RESTART: D:\Python\ex\ex13_4.py =====================
1 : 随机3组骰子值 :  [1, 4, 4]
2 : 随机3组骰子值 :  [1, 4, 6]
3 : 随机3组骰子值 :  [1, 5, 6]
4 : 随机3组骰子值 :  [3, 3, 4]
5 : 随机3组骰子值 :  [2, 3, 3]
```

5. 请重新设计 ch13_17.py，每执行一次即将输出的水果从列表内删除，直到 fruits 列表为空。(13-5 节)

```
===================== RESTART: D:\Python\ex\ex13_5.py =====================
执行前列表 :  ['苹果', '香蕉', '西瓜', '水蜜桃', '百香果']
删除 :  香蕉
目前列表 :  ['苹果', '西瓜', '水蜜桃', '百香果']
删除 :  百香果
目前列表 :  ['苹果', '西瓜', '水蜜桃']
删除 :  西瓜
目前列表 :  ['苹果', '水蜜桃']
删除 :  水蜜桃
目前列表 :  ['苹果']
删除 :  苹果
目前列表 :  []
```

6. 重新设计 ch13_17_1.py，产生 600 个 1 ～ 6 的值，最后以排序字典方式列出每个骰子值出现的次数。你的骰子值出现的次数可能和下列结果不同。(13-5 节)

```
===================== RESTART: D:/Python/ex/ex13_6.py =====================
1 : 83
2 : 105
3 : 106
4 : 95
5 : 123
6 : 88
```

7. 重新设计 ch13_18_1.py，取得威力彩号码，威力彩普通号与大乐透相同，但是特别号是 1 ～ 8 的数字，这个程序会先列出特别号，再将一般号码由小到大排列。(13-5 节)

```
===================== RESTART: D:\Python\ex\ex13_7.py =====================
第1000期威力彩号码
特别号:4
13 19 32 35 42 46
```

8. 请列出目前你所使用的 Python 版本（version, version_info）、平台、窗口版本、可执行文件路径。(13-7 节)

```
===================== RESTART: D:\Python\ex\ex13_8.py =====================
目前Python版本是:      3.7.1 (v3.7.1:260ec2c36a, Oct 20 2018, 14:05:16) [MSC v.1
915 32 bit (Intel)]
目前Python版本是:      sys.version_info(major=3, minor=7, micro=1, releaselevel=
'final', serial=0)
目前Python平台是:      win32
目前Python窗口版本是:  sys.getwindowsversion(major=10, minor=0, build=17134, pla
tform=2, service_pack='')
目前Python可执行文件路径 C:\Users\User\AppData\Local\Programs\Python\Python37-32
\pythonw.exe
```

9. 请输入字符串，本程序可以判断是不是 Python 关键词。(13-8 节)

```
===================== RESTART: D:\Python\ex\ex13_9.py =====================
输入字符串 : as
as 是关键词
输入字符串 : else
else 是关键词
输入字符串 : ak
ak 不是关键词
输入字符串 : q
q 不是关键词
```

10. 请重新设计 ch13_31.py，但是将年份和月份改为屏幕输入。(13-9 节)

```
===================== RESTART: D:\Python\ex\ex13_10.py =====================
请输入年份 : 2019
请输入月份 : 12
   December 2019
Mo Tu We Th Fr Sa Su
                   1
 2  3  4  5  6  7  8
 9 10 11 12 13 14 15
16 17 18 19 20 21 22
23 24 25 26 27 28 29
30 31
```

11. 扩充程序实例 ch13_50.py，多设计一个解密函数，将加密结果字符串解密。（13-11 节）

```
====================== RESTART: D:\Python\ex\ex13_11.py ======================
打印译码字典
 {'0': '}', '1': '~', '2': ' ', '3': '0', '4': '1', '5': '2', '6': '3', '7': '4'
, '8': '5', '9': '6', 'a': '7', 'b': '8', 'c': '9', 'd': 'a', 'e': 'b', 'f': 'c'
, 'g': 'd', 'h': 'e', 'i': 'f', 'j': 'g', 'k': 'h', 'l': 'i', 'm': 'j', 'n': 'k'
, 'o': 'l', 'p': 'm', 'q': 'n', 'r': 'o', 's': 'p', 't': 'q', 'u': 'r', 'v': 's'
, 'w': 't', 'x': 'u', 'y': 'v', 'z': 'w', 'A': 'x', 'B': 'y', 'C': 'z', 'D': 'A'
, 'E': 'B', 'F': 'C', 'G': 'D', 'H': 'E', 'I': 'F', 'J': 'G', 'K': 'H', 'L': 'I'
, 'M': 'J', 'N': 'K', 'O': 'L', 'P': 'M', 'Q': 'N', 'R': 'O', 'S': 'P', 'T': 'Q'
, 'U': 'R', 'V': 'S', 'W': 'T', 'X': 'U', 'Y': 'V', 'Z': 'W', '!': 'X', '"': 'Y'
, '#': 'Z', '$': '!', '%': '"', '&': '#', "'": '$', '(': '%', ')': '&', '*': "'"
, '+': '(', ',': ')', '-': '*', '.': '+', '/': ',', ':': '-', ';': '.', '<': '/'
, '=': ':', '>': ';', '?': '<', '@': '=', '[': '>', '\\': '?', ']': '@', '^': '['
, '_': '\\', '`': ']', '{': '^', '|': '_', '}': '`', '~': '{', ' ': '|'}
原始字符串  If the implementation is easy to explain, it may be a good idea.
加密字符串  Li2wkh2lpsohphqwdwlrq2lv2hdvB2wr2hAsodlq/2lw2pdB2eh2d2jrrg2lghd;
解密字符串  If the implementation is easy to explain, it may be a good idea.
```

12. 扩充程序实例 ch13_51.py，多设计一个解密函数，将加密结果字符串解密。（13-11 节）

```
====================== RESTART: D:\Python\ex\ex13_12.py ======================
打印译码字典
 {'0': 'O', '1': 'k', '2': ' ', '3': 'A', '4': '_', '5': 'K', '6': 'L', '7': 'w'
, '8': '(', '9': 't', 'a': '%', 'b': '!', 'c': 'j', 'd': '7', 'e': '0', 'f': 'E'
, 'g': '&', 'h': '2', 'i': '9', 'j': 'u', 'k': 'i', 'l': 'Y', 'm': 'T', 'n': 'm'
, 'o': '+', 'p': '{', 'q': 'N', 'r': ':', 's': 'R', 't': '5', 'u': '|', 'v': 'h'
, 'w': '8', 'x': 's', 'y': 'c', 'z': 'q', 'A': 'J', 'B': 'y', 'C': '}', 'D': 'M'
, 'E': '4', 'F': 'V', 'G': '6', 'H': 'f', 'I': ',', 'J': 'z', 'K': 'p', 'L': 'e'
, 'M': 'C', 'N': '?', 'O': '.', 'P': 'P', 'Q': ';', 'R': 'U', 'S': 'a', 'T': 'x'
, 'U': 'd', 'V': 'B', 'W': ']', 'X': '$', 'Y': '/', 'Z': '-', '!': 'H', '"': '"'
, '#': '[', '$': 'v', '%': '#', '&': 'r', "'": 'I', '(': 'W', ')': 'G', '*': 'g'
, '+': '@', ',': '`', '-': ')', '.': '\\', '/': '=', ':': '<', ';': 'Z', '<': '~
', '=': '^', '>': '3', '?': 'Q', '@': 'F', '[': 'n', '\\': '1', ']': 'D', '^': '
b', '_': '>', '`': '"', '{': 'S', '|': 'X', '}': 'l', '~': 'o', ' ': '*'}
原始字符串  If the implementation is easy to explain, it may be a good idea.
加密字符串  'H29vL2knK\LnL[9S9k~[2kx2LSxB29~2LTK\Sk[I2k92nSB2^L2S2*~~U2kULSO
解密字符串  If the implementation is easy to explain, it may be a good idea.
```

14

第 14 章

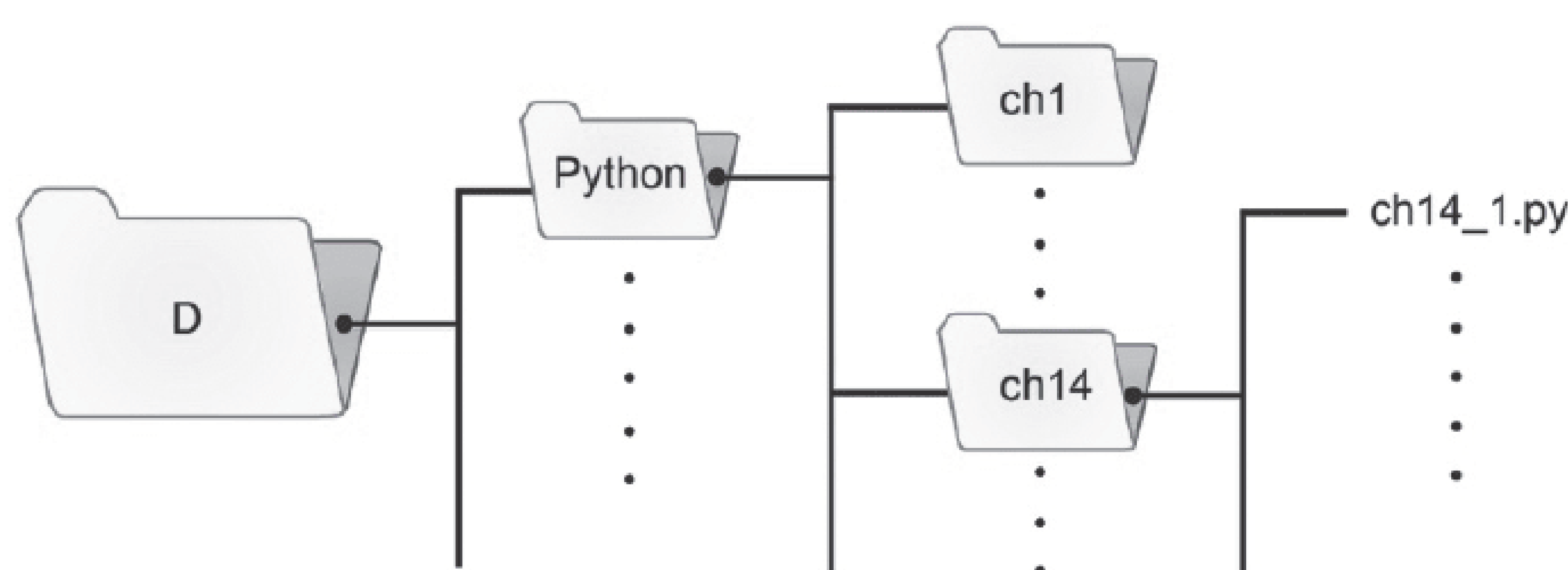
文件的读取与写入

本章摘要

本章将讲解使用 Python 处理 Windows 操作系统内文件的相关知识，例如，**文件路径的管理、文件的读取与写入、目录的管理、文件压缩与解压缩、认识编码规则与剪贴板**的相关应用。

14-1 文件夹与文件路径

有一个文件路径如下：



对于 ch14_1.py 而言，它的文件路径是：

```
D:\Python\ch14\ch14_1.py
```

对于 ch14_1.py 而言，它的目前工作**目录**（也可称**文件夹**）名称是：

```
D:\Python\ch14
```

14-1-1 绝对路径与相对路径

在操作系统中可以使用两种方式表达文件路径，下面是以 ch14_1.py 为例。

（1）**绝对路径**：路径从根目录开始表达，例如，若以 14-1 节的文件路径为例，它的绝对路径是：

```
D:\Python\ch14\ch14_1.py
```

（2）**相对路径**：是指相对于目前工作目录的路径，例如，若以 14-1 节的文件路径为例，若是目前工作目录是 D:\Python\ch14，它的相对路径是：

```
ch14_1.py
```

另外，在操作系统处理文件夹的概念中会使用两个特殊符号“.”和“..”，“.”指的是目前文件夹，“..”指的是上一层文件夹。但是在使用上，当指目前文件夹时也**可以省略**“.\”。所以使用“.\ch14_1.py”与“ch14_1.py”意义相同。

14-1-2 os 模块与 os.path 模块

在 Python 内有关文件路径的模块是 os，所以在本节实例最前面均需导入此模块。

```
import os                    # 导入 os 模块
```

在 os 模块内有另一个常用**模块 os.path**，14-1 节主要是使用这两个模块的方法，讲解与文件路径有关的文件夹知识，由于 os.path 是在 os 模块内，所以导入 os 模块后不用再导入 os.path 模块。

14-1-3　取得目前工作目录 os.getcwd( )

os 模块内的 getcwd( ) 可以取得目前工作目录。

程序实例 ch14_1.py：列出目前工作目录。

```
1  # ch14_1.py
2  import os
3
4  print(os.getcwd())              # 列出目前工作目录
```

执行结果

```
===================== RESTART: D:\Python\ch14\ch14_1.py =====================
D:\Python\ch14
```

14-1-4　取得绝对路径 os.path.abspath

os.path 模块内的 abspath(path) 会返回 path 的绝对路径，通常可以使用这个方法将文件或文件夹的相对路径转成绝对路径。

程序实例 ch14_2.py：取得绝对路径的应用。

```
1  # ch14_2.py
2  import os
3
4  print(os.path.abspath('.'))            # 列出目前工作目录的绝对路径
5  print(os.path.abspath('..'))           # 列出上一层工作目录的绝对路径
6  print(os.path.abspath('ch14_2.py'))    # 列出目前文件的绝对路径
```

执行结果

```
===================== RESTART: D:\Python\ch14\ch14_2.py =====================
D:\Python\ch14
D:\Python
D:\Python\ch14\ch14_2.py
```

14-1-5　返回特定路段相对路径 os.path.relpath( )

os.path 模块内的 relpath(path, start) 会返回从 start 到 path 的相对路径，如果省略 start，则返回目前工作目录至 path 的相对路径。

程序实例 ch14_3.py：返回特定路段相对路径的应用。

```
1  # ch14_3.py
2  import os
3
4  print(os.path.relpath('D:\\'))              # 列出目前工作目录至D:\的相对路径
5  print(os.path.relpath('D:\\Python\\ch13'))  # 列出目前工作目录至特定path的相对路径
6  print(os.path.relpath('D:\\', 'ch14_3.py')) # 列出目前文件至D:\的相对路径
```

执行结果

```
===================== RESTART: D:\Python\ch14\ch14_3.py =====================
..\..
..\ch13
..\..\..
```

14-1-6 检查路径方法 exist/isabs/isdir/isfile

下列是常用的 os.path 模块方法。

exist(path)：如果 path 的文件或文件夹存在，返回 True，否则返回 False。

isabs(path)：如果 path 的文件或文件夹是绝对路径，返回 True，否则返回 False。

isdir(path)：如果 path 是文件夹，返回 True，否则返回 False。

isfile(path)：如果 path 是文件，返回 True，否则返回 False。

程序实例 ch14_4.py：检查路径方法的应用。

```
1  # ch14_4.py
2  import os
3
4  print("文件或文件夹存在 = ", os.path.exists('ch14'))
5  print("文件或文件夹存在 = ", os.path.exists('D:\\Python\\ch14'))
6  print("文件或文件夹存在 = ", os.path.exists('ch14_4.py'))
7  print(" --- ")
8
9  print("是绝对路径 = ", os.path.isabs('ch14_4.py'))
10 print("是绝对路径 = ", os.path.isabs('D:\\Python\\ch14\\ch14_4.py'))
11 print(" --- ")
12
13 print("是文件夹 = ", os.path.isdir('D:\\Python\\ch14\\ch14_4.py'))
14 print("是文件夹 = ", os.path.isdir('D:\\Python\\ch14'))
15 print(" --- ")
16
17 print("是文件 = ", os.path.isfile('D:\\Python\\ch14\\ch14_4.py'))
18 print("是文件 = ", os.path.isfile('D:\\Python\\ch14'))
```

执行结果

```
===================== RESTART: D:\Python\ch14\ch14_4.py =====================
文件或文件夹存在 =  False
文件或文件夹存在 =  True
文件或文件夹存在 =  True
 ---
是绝对路径 =  False
是绝对路径 =  True
 ---
是文件夹 =  False
是文件夹 =  True
 ---
是文件 =  True
是文件 =  False
>>>
```

14-1-7 文件与目录的操作 mkdir/rmdir/remove/chdir

这几个方法是在 os 模块内，建议执行下列操作前先用 os.path.exists( ) 检查路径是否存在。

mkdir(path)：建立 path 目录。

rmdir(path)：删除 path 目录，限制只能是空的目录。如果要删除底下有文件的目录需参考 14-5-7 节。

remove(path)：删除 path 文件。

chdir(path)：将目前工作文件夹改至 path。

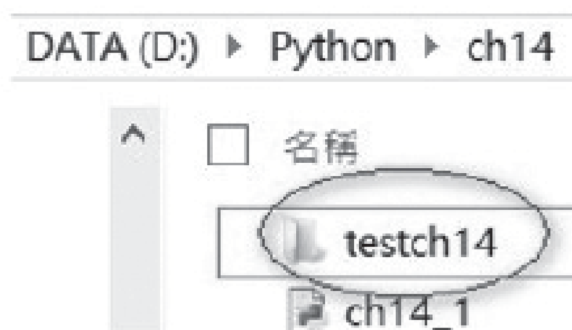
程序实例 ch14_5.py：使用 mkdir 建立文件夹的应用。

```
1  # ch14_5.py
2  import os
3
4  mydir = 'testch14'
5  # 如果mydir不存在就建立此文件夹
6  if os.path.exists(mydir):
7      print("已经存在 %s " % mydir)
8  else:
9      os.mkdir(mydir)
10     print("建立 %s 文件夹成功" % mydir)
```

执行结果

```
===================== RESTART: D:\Python\ch14\ch14_5.py =====================
建立 testch14 文件夹成功
>>>
```

下列是验证 testch14 建立成功的画面。



程序实例 ch14_6.py：使用 rmdir 删除文件夹的应用。

```
1  # ch14_6.py
2  import os
3
4  mydir = 'testch14'
5  # 如果mydir存在就删除此文件夹
6  if os.path.exists(mydir):
7      os.rmdir(mydir)
8      print("删除 %s 文件夹成功" % mydir)
9  else:
10     print("%s 文件夹不存在" % mydir)
```

执行结果

```
===================== RESTART: D:\Python\ch14\ch14_6.py =====================
testch14 文件夹不存在
>>>
```

程序实例 ch14_7.py：删除指定 path 文件的应用。

```
1  # ch14_7.py
2  import os
3
4  myfile = 'test.py'
5  # 如果myfile存在就删除此文件
6  if os.path.exists(myfile):
7      os.remove(myfile)
8      print("删除 %s 文件成功" % myfile)
9  else:
10     print("%s 文件不存在" % myfile)
```

执行结果 下列分别是删除文件不存在（左边）和存在（右边）的执行结果。

```
===================== RESTART: D:\Python\ch14\ch14_7.py
test.py 文件不存在
>>>
```

```
===================== RESTART: D:/Python/ch14/test.py
删除 test.py 文件成功
>>>
```

程序实例 ch14_8.py：更改目前工作文件夹，然后再返回原先工作文件夹。

```
13     print("建立 %s 文件夹成功" % newdir)
14
15 # 将目前工作文件夹改至newdir
16 os.chdir(newdir)
17 print("列出最新工作文件夹 ", os.getcwd())
18
19 # 将目前工作文件夹返回
20 os.chdir(currentdir)
21 print("列出返回工作文件夹 ", currentdir)
```

```
13         print("建立 %s 文件夹成功" % newdir)
14
15   # 将目前工作文件夹改至newdir
16   os.chdir(newdir)
17   print("列出最新工作文件夹 ", os.getcwd())
18
19   # 将目前工作文件夹返回
20   os.chdir(currentdir)
21   print("列出返回工作文件夹 ", currentdir)
```

执行结果

```
==================== RESTART: D:\Python\ch14\ch14_8.py ====================
列出目前工作文件夹  D:\Python\ch14
已经存在 D:\Python
列出最新工作文件夹  D:\Python
列出返回工作文件夹  D:\Python\ch14
>>>
```

14-1-8　返回文件路径 os.path.join( )

这个方法可以将 os.path.join( ) 参数内的字符串结合为一个文件路径，参数可以有两个到多个。

程序实例 ch14_9.py：os.path.join( ) 方法的应用，这个程序会分别用 2、3、4 个参数测试这个方法。

```
1   # ch14_9.py
2   import os
3
4   print(os.path.join('D:\\', 'Python', 'ch14', 'ch14_9.py'))    # 4个参数
5   print(os.path.join('D:\\Python', 'ch14', 'ch14_9.py'))        # 3个参数
6   print(os.path.join('D:\\Python\\ch14', 'ch14_9.py'))          # 2个参数
```

执行结果

```
==================== RESTART: D:\Python\ch14\ch14_9.py ====================
D:\Python\ch14\ch14_9.py
D:\Python\ch14\ch14_9.py
D:\Python\ch14\ch14_9.py
```

程序实例 ch14_10.py：使用 for 循环将一个列表内的文件与一个路径结合。

```
1   # ch14_10.py
2   import os
3
4   files = ['ch14_1.py', 'ch14_2.py', 'ch14_3.py']
5   for file in files:
6       print(os.path.join('D:\\Python\\ch14', file))
```

执行结果

```
==================== RESTART: D:\Python\ch14\ch14_10.py ====================
D:\Python\ch14\ch14_1.py
D:\Python\ch14\ch14_2.py
D:\Python\ch14\ch14_3.py
```

14-1-9　获得特定文件的大小 os.path.getsize( )

这个方法可以获得特定文件的大小。

程序实例 ch14_11.py：获得 ch14_1.py 的文件大小，从执行结果可以知道是 92B。

```
1  # ch14_11.py
2  import os
3
4  # 如果文件在目前工作目录下可以省略路径
5  print(os.path.getsize("ch14_1.py"))
6  print(os.path.getsize("D:\\Python\\ch14\\ch14_1.py"))
```

执行结果

```
==================== RESTART: D:\Python\ch14\ch14_11.py ====================
92
92
```

14-1-10　获得特定工作目录的内容 os.listdir( )

这个方法将以列表方式列出特定工作目录的内容。

程序实例 ch14_12.py：以两种方式列出 D:\Python\ch14 的工作目录内容。

```
1  # ch14_12.py
2  import os
3
4  print(os.listdir("D:\\Python\\ch14"))
5  print(os.listdir("."))                    # 这代表目前工作目录
```

执行结果

```
==================== RESTART: D:/Python/ch14/ch14_12.py ====================
['ch14_1.py', 'ch14_10.py', 'ch14_11.py', 'ch14_12.py', 'ch14_2.py', 'ch14_3.py'
, 'ch14_4.py', 'ch14_5.py', 'ch14_6.py', 'ch14_7.py', 'ch14_8.py', 'ch14_9.py',
'testch14']
['ch14_1.py', 'ch14_10.py', 'ch14_11.py', 'ch14_12.py', 'ch14_2.py', 'ch14_3.py'
, 'ch14_4.py', 'ch14_5.py', 'ch14_6.py', 'ch14_7.py', 'ch14_8.py', 'ch14_9.py',
'testch14']
>>>
```

程序实例 ch14_13.py：列出特定工作目录所有文件的大小。

```
1  # ch14_13.py
2  import os
3
4  totalsizes = 0
5  print("列出D:\\Python\\ch14工作目录的所有文件")
6  for file in os.listdir('D:\\Python\\ch14'):
7      print(file)
8      totalsizes += os.path.getsize(os.path.join('D:\\Python\\ch14', file))
9
10 print("全部文件大小是 = ", totalsizes)
```

执行结果

```
==================== RESTART: D:\Python\ch14\ch14_13.py ====================
列出D:\Python\ch14工作目录的所有文件
ch14_1.py
ch14_10.py
ch14_11.py
ch14_12.py
ch14_13.py
ch14_2.py
ch14_3.py
ch14_4.py
ch14_5.py
ch14_6.py
ch14_7.py
ch14_8.py
ch14_9.py
全部文件大小是 =  3631
>>>
```

14-1-11　获得特定工作目录内容 glob

Python 内还有一个模块 glob 可用于列出特定工作目录内容，当导入这个模块后，可以使用 glob 方法获得特定工作目录的内容，这个方法最大的特点是可以使用通配符“*”，例如，可用“*.txt”获得所有 txt 扩展名的文件，“?”可以匹配任意字符，“[abc]”必须是 abc 字符。更多应用可参考下列实例。

程序实例 ch14_14.py：方法 1 是列出所有工作目录下的文件，方法 2 是列出以 ch14_1 开头的扩展名是 py 的文件，方法 3 是列出以 ch14_2 开头的所有文件。

```
 1  # ch14_14.py
 2  import glob
 3
 4  print("方法1:列出\\Python\\ch14工作目录的所有文件")
 5  for file in glob.glob('D:\\Python\\ch14\*.*'):
 6      print(file)
 7
 8  print("方法2:列出目前工作目录的特定文件")
 9  for file in glob.glob('ch14_1*.py'):
10      print(file)
11
12  print("方法3:列出目前工作目录的特定文件")
13  for file in glob.glob('ch14_2*.*'):
14      print(file)
```

执行结果

```
===================== RESTART: D:\Python\ch14\ch14_14.py =====================
方法1:列出\Python\ch14工作目录的所有文件
D:\Python\ch14\ch14_1.py
D:\Python\ch14\ch14_10.py
D:\Python\ch14\ch14_11.py
D:\Python\ch14\ch14_12.py
D:\Python\ch14\ch14_13.py
D:\Python\ch14\ch14_14.py
D:\Python\ch14\ch14_2.py
D:\Python\ch14\ch14_3.py
D:\Python\ch14\ch14_4.py
D:\Python\ch14\ch14_5.py
D:\Python\ch14\ch14_6.py
D:\Python\ch14\ch14_7.py
D:\Python\ch14\ch14_8.py
D:\Python\ch14\ch14_9.py
方法2:列出目前工作目录的特定文件
ch14_1.py
ch14_10.py
ch14_11.py
ch14_12.py
ch14_13.py
ch14_14.py
方法3:列出目前工作目录的特定文件
ch14_2.py
>>>
```

14-1-12　遍历目录树 os.walk( )

在 os 模块内提供了一个 os.walk( ) 方法可以遍历目录树，这个方法每次执行循环时将返回以下 3 个值。

（1）目前工作目录名称（dirName）。

（2）目前工作目录下的子目录列表（sub_dirNames）。

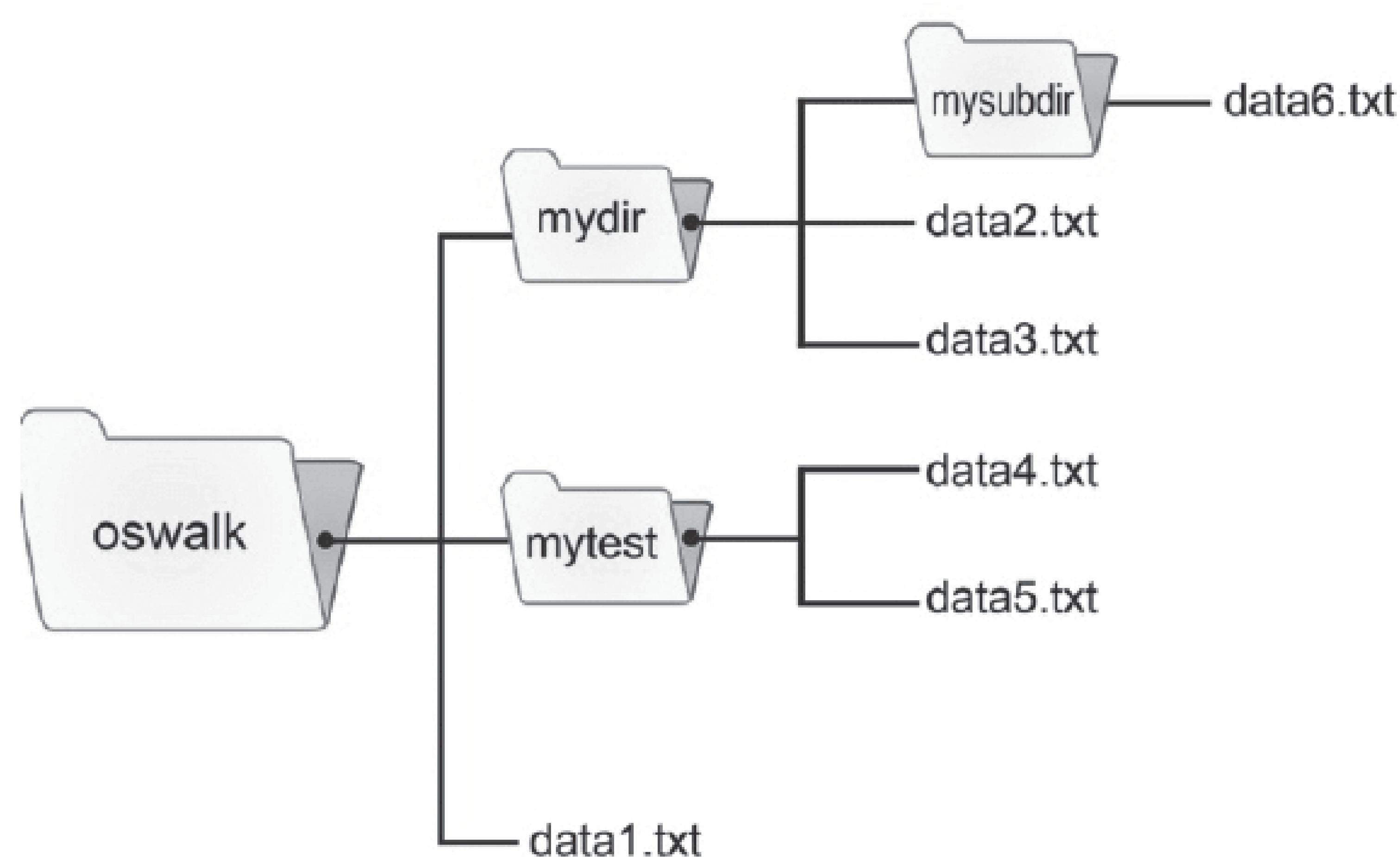
（3）目前工作目录下的文件列表（fileNames）。

下面是语法格式。

```
for dirName, sub_dirNames, fileNames in os.walk(目录路径):
    程序区块
```

上述 dirName, sub_dirNames, fileNames 名称可以自行命名，顺序则不可以更改，至于**目录路径**可以使用绝对地址或相对地址，**可以使用 os.walk ('.') 代表目前工作目录**。

程序实例 ch14_14_1.py：在范例 D:\Python\ch14 目录下列有一个 oswalk 目录，此目录内容如下。



本程序将遍历此 oswalk 目录，同时列出内容。

```
1  # ch14_14_1.py
2  import os
3
4  for dirName, sub_dirNames, fileNames in os.walk('oswalk'):
5      print("目前工作目录名称:    ", dirName)
6      print("目前子目录名称列表: ", sub_dirNames)
7      print("目前文件名列表:    ", fileNames, "\n")
```

执行结果

```
==================== RESTART: D:\Python\ch14\ch14_14_1.py ====================
目前工作目录名称:    oswalk
目前子目录名称列表:  ['mydir', 'mytest']
目前文件名列表:     ['data1.txt']

目前工作目录名称:    oswalk\mydir
目前子目录名称列表:  ['mysubdir']
目前文件名列表:     ['data2.txt', 'data3.txt']

目前工作目录名称:    oswalk\mydir\mysubdir
目前子目录名称列表:  []
目前文件名列表:     ['data6.txt']

目前工作目录名称:    oswalk\mytest
目前子目录名称列表:  []
目前文件名列表:     ['data4.txt', 'data5.txt']

>>>
```

从上述执行结果可以看到，os.walk( ) 将遍历指定目录下的子目录，同时返回子目录列表和文件列表，如果所返回的子目录列表是 [ ] 代表其下没有子目录。

14-2 读取文件

Python 处理读取或写入文件首先需将文件打开，然后可以接收一次读取所有文件内容或是一行一行读取文件内容。Python 可以使用 open( ) 函数打开文件，文件打开后会返回文件对象，未来可用读取此文件对象方式读取文件内容，更多有关 open( ) 函数的内容可参考 4-3-1 节。

14-2-1　读取整个文件 read( )

文件打开后，可以使用 read( ) 读取所打开的文件。使用 read( ) 读取时，所有的文件内容将以一个字符串方式被读取然后存入字符串变量内，未来只要打印此字符串变量相当于可以打印整个文件内容。

在本书文件夹的 ch14 文件夹中有下列 ch14_15.txt 文件。

```
DeepMind
DeepStone
Deep Learning
```

程序实例 ch14_15.py：读取 ch14_15.txt 文件然后输出，请读者留意程序第 7 行，笔者使用打印一般变量的方式就打印了整个文件。

```
1  # ch14_15.py
2
3  fn = 'ch14_15.txt'              # 设置要打开的文件
4  file_Obj =  open(fn)            # 用预设mode=r打开文件,返回调用对象file_Obj
5  data = file_Obj.read()          # 读取文件到变量data
6  file_Obj.close()                # 关闭文件对象
7  print(data)                     # 输出变量data相当于输出文件
```

执行结果

```
==================== RESTART: D:\Python\ch14\ch14_15.py ====================
DeepMind
DeepStone
Deep Learning

>>>
```

上述语句使用 open( ) 打开文件时，建议使用 close( ) 将文件关闭，可参考第 6 行。若是没有关闭也许未来文件内容会有不可预期的损害。

另外，上述程序第 3 和 4 行所打开的文件 ch14_15.txt 没有文件路径，这表示这个文件需与程序文件在相同的工作目录，否则会有找不到这个文件的情况发生。当然设计程序时，也可以在第 3 行直接配置文件的绝对路径，如下所示。

```
D:\Python\ch14\ch14_15.txt
```

如果这样，就不必担心数据文件 ch14_15.txt 与程序文件 ch14_15.py 是否在相同目录了。

14-2-2　with 关键词

其实 Python 提供了一个关键词 with，应用在打开文件与建立文件对象时使用方式如下。

```
with open(要打开的文件) as 文件对象:
    相关系列指令
```

真正懂 Python 的使用者都是使用这种方式打开文件，最大的特点是可以不必在程序中关闭文件，with 指令会在结束不需要此文件时自动将它关闭，文件经“with open( ) as 文件对象”打开后会有一个文件对象，就可以使用 read( ) 读取此文件对象的内容了。

程序实例 ch14_16.py：使用 with 关键词重新设计 ch14_15.py。

```
1  # ch14_16.py
2
3  fn = 'ch14_15.txt'              # 设置要打开的文件
4  with open(fn) as file_Obj:      # 用默认mode=r打开文件,返回调用对象file_Obj
5      data = file_Obj.read()      # 读取文件到变量data
6      print(data)                 # 输出变量data相当于输出文件
```

执行结果 与 ch14_15.py 相同。

由于整个文件是以字符串方式被读取与存储，所以打印字符串时最后一行的空白行也将显示出来，不过可以使用 rstrip( ) 将 data 字符串变量（文件）末端的空格符删除。

程序实例 ch14_17.py：重新设计 ch14_16.py，但是删除文件末端的空白。

```
1  # ch14_17.py
2
3  fn = 'ch14_15.txt'              # 设置要打开的文件
4  with open(fn) as file_Obj:      # 用默认mode=r打开文件,返回调用对象file_Obj
5      data = file_Obj.read()      # 读取文件到变量data
6      print(data.rstrip())        # 输出变量data相当于输出文件,同时删除末端字符
```

执行结果

```
===================== RESTART: D:\Python\ch14\ch14_17.py =====================
DeepMind
DeepStone
Deep Learning
>>>
```

由执行结果可以看到文件末端不再有空白行了。

14-2-3　逐行读取文件内容

在 Python 中若想逐行读取文件内容，可以使用下列循环。

for **line** in file_Obj:　# **line** 和 **fileObj** 可以自行取名 ,**file_Obj** 是文件对象

循环相关系列指令

程序实例 ch14_18.py：逐行读取和输出文件。

```
1  # ch14_18.py
2
3  fn = 'ch14_15.txt'              # 设置要打开的文件
4  with open(fn) as file_Obj:      # 用默认mode=r打开文件,返回调用对象file_Obj
5      for line in file_Obj:       # 逐行读取文件到变量line
6          print(line)             # 输出变量line相当于输出一行
```

执行结果

```
===================== RESTART: D:\Python\ch14\ch14_18.py =====================
DeepMind

DeepStone

Deep Learning

>>>
```

因为以记事本编辑的 ch14_15.txt 文本文件每行末端有换行符号，同时 print( ) 在输出时也有一

个换行输出的符号，所以才会得到上述每行输出后有空一行的结果。

程序实例 ch14_19.py：重新设计 ch14_18.py，但是删除每行末端的换行符号。

```
1  # ch14_19.py
2
3  fn = 'ch14_15.txt'              # 设置要打开的文件
4  with open(fn) as file_Obj:      # 用默认mode=r打开文件,返回调用对象file_Obj
5      for line in file_Obj:       # 逐行读取文件到变量line
6          print(line.rstrip())    # 输出变量line相当于输出一行,同时删除末端字符
```

执行结果

```
===================== RESTART: D:\Python\ch14\ch14_19.py =====================
DeepMind
DeepStone
Deep Learning
>>>
```

14-2-4 逐行读取使用 readlines( )

使用 with 关键词配合 open( ) 时，所打开的文件对象目前只在 with 区块内使用，适用在特别是想要遍历此文件对象时。Python 另外有一个方法 readlines( ) 可以采用逐行读取方式，一次读取全部 txt 的内容，同时以列表方式存储，另一个特点是读取时每行的换行字符都会存储在列表内。当然更重要的是可以在 with 区块外遍历原先文件对象内容。

在本书文件夹的 ch14 文件夹有下列 ch14_20.txt 文件。

```
Ming-Chi Institute of Technology
Ming-Chi University of Technology
I Love Ming-Chi

```

程序实例 ch14_20.py：使用 readlines( ) 逐行读取 ch14_20.txt，存入列表，然后打印此列表的结果。

```
1  # ch14_20.py
2
3  fn = 'ch14_20.txt'            # 设置要打开的文件
4  with open(fn) as file_Obj:    # 用默认mode=r打开文件,返回调用对象file_Obj
5      obj_list = file_Obj.readlines()  # 每次读一行
6
7  print(obj_list)               # 打印列表
```

执行结果

```
===================== RESTART: D:\Python\ch14\ch14_20.py =====================
['Ming-Chi Institute of Technology\n', 'Ming-Chi University of Technology\n', 'I
 Love Ming-Chi\n', '\n']
>>>
```

由上述执行结果可以看到，在 txt 文件中的换行字符也出现在列表元素内。

程序实例 ch14_21.py：逐行输出 ch14_20.py 所保存的列表内容。

```
1  # ch14_21.py
2
3  fn = 'ch14_20.txt'             # 设置要打开的文件
4  with open(fn) as file_Obj:     # 用默认mode=r打开文件,返回调用对象file_Obj
5      obj_list = file_Obj.readlines()  # 每次读一行
6
7  for line in obj_list:
8      print(line.rstrip())       # 打印列表
```

执行结果

```
===================== RESTART: D:\Python\ch14\ch14_21.py =====================
Ming-Chi Institute of Technology
Ming-Chi University of Technology
I Love Ming-Chi
```

14-2-5　数据组合

Python 的多功能用途，可以让我们很轻松地组合数据，例如，可以将原先分成 3 行显示的数据，以一个空格或不空格方式显示。

程序实例 ch14_22.py：重新设计 ch14_21.py，将分成 3 行显示的数据用 1 行显示。

```
1  # ch14_22.py
2
3  fn = 'ch14_20.txt'              # 设置要打开的文件
4  with open(fn) as file_Obj:      # 用默认mode=r打开文件,返回调用对象file_Obj
5      obj_list = file_Obj.readlines()   # 每次读一行
6
7  str_Obj = ''                    # 先设为空字符串
8  for line in obj_list:           # 将各行字符串存入
9      str_Obj += line.rstrip()
10
11 print(str_Obj)                  # 打印文件字符串
```

执行结果

```
===================== RESTART: D:\Python\ch14\ch14_22.py =====================
Ming-Chi Institute of TechnologyMing-Chi University of TechnologyI Love Ming-Chi
>>>
```

14-2-6　字符串的替换

使用 Word 进行字处理时常常会使用**查找 / 替换**功能，Python 也有这个方法，可以使用新字符串取代旧字符串。

字符串对象 .replace（旧字符串 , 新字符串）　　# 在字符串对象内，**新字符串将取代旧字符串**

程序实例 ch14_23.py：重新设计 ch14_21.py，但是将“工专”改为“科大”。

```
1  # ch14_23.py
2
3  fn = 'ch14_20.txt'                        # 设置要打开的文件
4  with open(fn) as file_Obj:                # 返回调用物件file_Obj
5      data = file_Obj.read()                # 读取文件到变量data
6      new_data = data.replace('工专', '科大') # 新变量存储
7      print(new_data.rstrip())              # 输出文件
```

执行结果

```
===================== RESTART: D:\Python\ch14\ch14_23.py =====================
Ming-Chi Institute of Technology
Ming-Chi University of Technology
I Love Ming-Chi
>>>
```

14-2-7 数据的查找

使用 Word 软件时也常会有查找功能，使用 Python 这类工作将变得相对简单。在本书文件夹的 ch14 文件夹有下列 sse.txt 文件。


sse -记事本
档案(F) 编辑(E) 格式(O) 检视(V) 说明(H)
Silicon Stone Education is a world leader in education-based certification exams and practice test solutions for academic institutions, workforce and corporate technology markets, delivered through an expansive network of over 250+ Silicon Stone Education Authorized testing sites worldwide in America Asia and Europe.

程序实例 ch14_24.py：数据查找的应用。这个程序会读取 sse.txt 文件，然后要求输入要查找的字符串，最后会响应此字符串是否在 sse.txt 文件中。

```
1   # ch14_24.py
2
3   fn = 'sse.txt'                          # 设置要打开的文件
4   with open(fn) as file_Obj:              # 用默认mode=r打开文件,返回调用对象file_Obj
5       obj_list = file_Obj.readlines()     # 每次读一行
6
7   str_Obj = ''                            # 先设为空字符串
8   for line in obj_list:                   # 将各行字符串存入
9       str_Obj += line.rstrip()
10
11  findstr = input("请输入要查找字符串 = ")
12  if findstr in str_Obj:                  # 查找文件是否有要寻找字符串
13      print(" 查找%s 字符串存在 %s 文件中" % (findstr, fn))
14  else:
15      print(" 查找%s 字符串不存在 %s 文件中" % (findstr, fn))
```

执行结果

```
===================== RESTART: D:\Python\ch14\ch14_24.py =====================
请输入要查找字符串 = Stone
查找 Stone 字符串存在 sse.txt 文件中
>>>
===================== RESTART: D:\Python\ch14\ch14_24.py =====================
请输入要查找字符串 = Deep
查找 Deep 字符串不存在 sse.txt 文件中
>>>
```

14-2-8 数据查找使用 find( )

对于字符串的使用，Python 提供了一个 find( ) 方法，这个方法除了可以执行数据查找外，如果查找到数据还会返回数据的索引位置，如果没有找到则返回 -1。

index = S.find(sub[, start[, end]]) # S 代表被查找的字符串，sub 是要查找字符串

index 是如果查找到时返回的索引值，start 和 end 代表可以被查找字符串的区间，若是省略表示全部查找，如果没有找到则返回 -1 给 index。

程序实例 ch14_25.py：重新设计 ch14_24.py，当查找到字符串时同时列出字符串所在索引的位置。

```
1  # ch14_25.py
2
3  fn = 'sse.txt'                         # 设置要打开的文件
4  with open(fn) as file_Obj:             # 用默认mode=r打开文件,返回调用对象file_Obj
5      obj_list = file_Obj.readlines()    # 每次读一行
6
7  str_Obj = ''                           # 先设为空字符串
8  for line in obj_list:                  # 将各行字符串存入
9      str_Obj += line.rstrip()
10
11 findstr = input("请输入要查找字符串 = ")
12 index = str_Obj.find(findstr)          # 查找findstr字符串是否存在
13 if  index >= 0:                        # 查找文件是否有要寻找字符串
14     print("查找 %s 字符串存在 %s 文件中" % (findstr, fn))
15     print("在索引 %s 位置出现" % index)
16 else:
17     print("查找 %s 字符串不存在 %s 文件中" % (findstr, fn))
```

执行结果

```
===================== RESTART: D:\Python\ch14\ch14_25.py =====================
请输入要查找字符串 = sse
查找 sse 字符串不存在 sse.txt 文件中
>>>
===================== RESTART: D:\Python\ch14\ch14_25.py =====================
请输入要查找字符串 = Stone
查找 Stone 字符串存在 sse.txt 文件中
在索引 8 位置出现
>>>
```

14-2-9　数据查找 rfind( )

rfind( ) 方法可以查找特定子字符串最后一次出现的位置，它的语法如下。

```
index = S.rfind(sub[, start[, end]]) # S 代表被查找字符串， sub 是要
                                       查找子字符串
```

index 是如果查找到时返回的索引值，start 和 end 代表可以被查找字符串的区间，若是省略表示全部查找，如果没有找到则返回 -1 给 index。

程序实例 ch14_25_1.py：在字符串中查找子字符串的应用。

```
1  # ch14_25_1.py
2  msg = '''CIA Mark told CIA Linda that the secret USB
3  had given to CIA Peter'''
4  print("CIA最后出现位置: ", msg.rfind("CIA",0,len(msg)))
```

执行结果

```
===================== RESTART: D:\Python\ch14\ch14_25_1.py =====================
CIA最后出现位置:  57
```

上述第 4 行 rifnd( ) 第 2 个参数 0 代表从头开始查找，第 3 个参数 len(msg) 可以计算原始字符串长度代表查找全部字符串。

14-2-10　分批读取文件数据

在真实的文件读取应用中，如果文件很大时，可能要分批读取文件数据，下面是分批读取文件的应用。

程序实例 ch14_25_2.py：用一次读取 100 字符的方式，读取 sse.txt 文件。

```
1  # ch14_25_2.py
2
3  fn = 'sse.txt'                # 设置要打开的文件
4  chunk = 100
5  msg = ''
6  with open(fn) as file_Obj:   # 用默认mode=r打开文件,返回调用对象file_Obj
7      while True:
8          txt = file_Obj.read(chunk)       # 一次读取chunk数量
9          if not txt:
10             break
11         msg += txt
12 print(msg)
```

执行结果

```
==================== RESTART: D:/Python/ch14/ch14_25_2.py ====================
Silicon Stone Education is a world leader in education-based
 certification exams and practice test solutions for academic
institutions, workforce and corporate technology markets,
delivered through an expansive network of over 250+ Silicon
Stone Education Authorized testing sites worldwide in America,
Asia and Europe.
```

14-3 写入文件

程序设计时一定会碰上要求将执行结果保存起来，此时就可以将执行结果存入文件内。

14-3-1 将执行结果写入空的文件内

打开文件函数 open( ) 使用时默认是 mode='r'，即读取文件模式，因此如果打开文件是供读取可以省略 mode='r'。若是要供写入，那么就要设置写入模式 mode='w'，程序设计时可以省略 mode，直接在 open( ) 函数内输入 'w'。如果所打开的文件可以读取或写入，可以使用 'r+'。如果所打开的文件不存在，open( ) 会建立该文件对象，如果所打开的文件已经存在，原文件内容将被清空。

至于输出到文件可以使用 write( ) 方法，语法格式如下。

len = 文件对象 .write(要输出数据)　　　　# 可将数据输出到文件对象

上述方法会返回输出数据的数据长度。

程序实例 ch14_26.py：输出数据到文件的应用。

```
1  # ch14_26.py
2  fn = 'out14_26.txt'
3  string = 'I love Python.'
4
5  with open(fn, 'w') as file_Obj:
6      file_Obj.write(string)
```

执行结果 这个程序在执行时在 Python Shell 窗口中看不到结果，必须到 ch14 工作目录下查看所建的 out14_26.txt 文件，同时打开可以得到下列结果。



程序实例 ch14_26_1.py：重新设计 ch14_26.py，这个程序会返回数据长度。

```
1  # ch14_26_1.py
2  fn = 'out14_26.txt'
3  string = 'I love Python.'
4
5  with open(fn, 'w') as file_Obj:
6      print(file_Obj.write(string))
```

执行结果

```
==================== RESTART: D:/Python/ch14/ch14_26_1.py ====================
14
```

14-3-2　写入数值资料

write( ) 输出时无法输出数值数据，可参考下列错误范例。

程序实例 ch14_27.py：使用 write( ) 输出数值数据产生错误的实例。

```
1  # ch14_27.py
2  fn = 'out14_27.txt'
3  x = 100
4
5  with open(fn, 'w') as file_Obj:
6      file_Obj.write(x)                # 直接输出数值x产生错误
```

执行结果

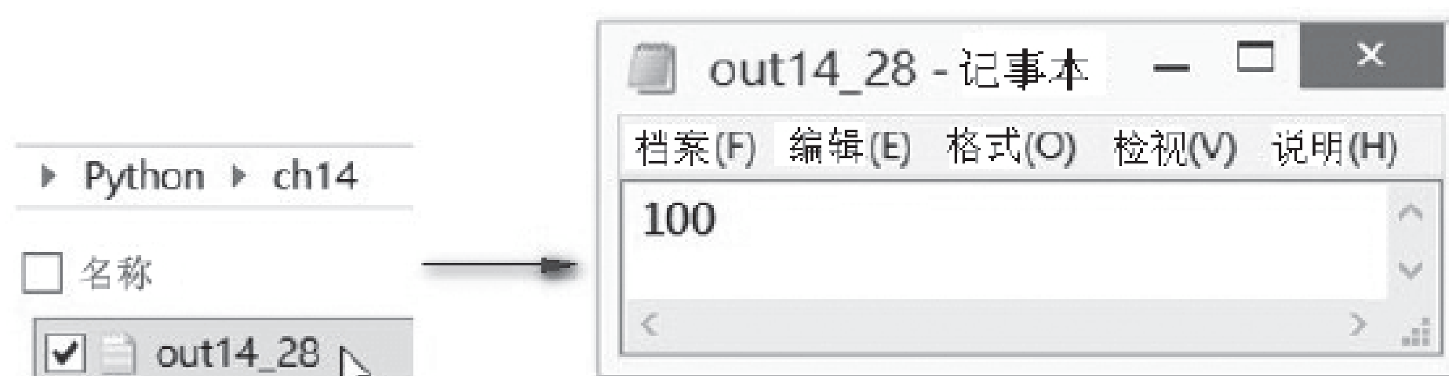
```
===================== RESTART: D:\Python\ch14\ch14_27.py =====================
Traceback (most recent call last):
  File "D:\Python\ch14\ch14_27.py", line 6, in <module>
    file_Obj.write(x)                # 直接输出数值x产生错误
TypeError: write() argument must be str, not int
>>>
```

如果想要使用 write( ) 将数值数据输出，必须使用 str( ) 将数值数据转成字符串数据。

程序实例 ch14_28.py：将数值数据转成字符串数据输出的实例。

```
1  # ch14_28.py
2  fn = 'out14_28.txt'
3  x = 100
4
5  with open(fn, 'w') as file_Obj:
6      file_Obj.write(str(x))      # 使用str(x)输出
```

执行结果 这个程序执行时在 Python Shell 窗口中看不到结果，必须到 ch14 工作目录下查看所建的 out14_28.txt 文件，同时打开可以得到下列结果。



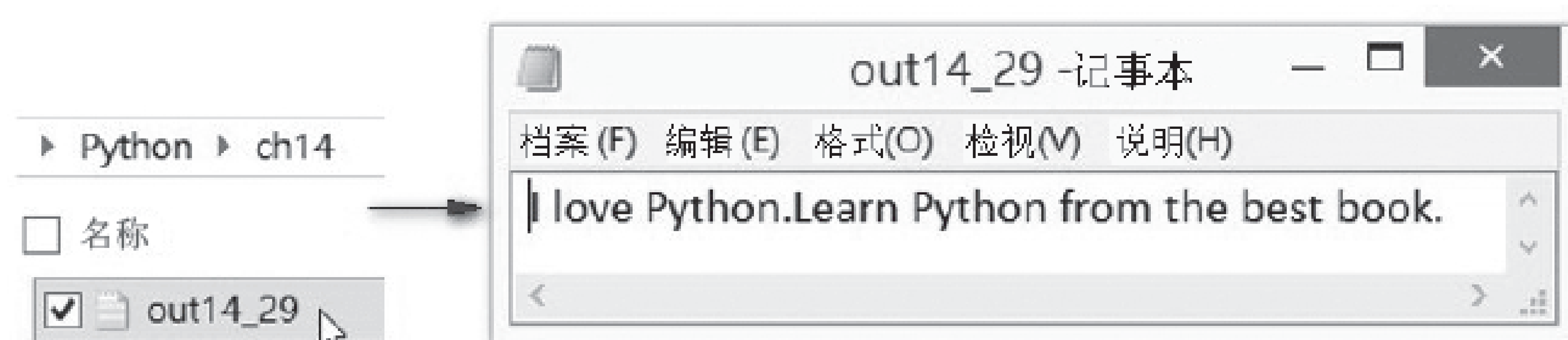
14-3-3 输出多行数据的实例

如果多行数据输出到文件中，设计程序时需留意各行间的换行符号问题，write( ) 不会主动在行的末端加上换行符号，如果有需要需自己处理。

程序实例 ch14_29.py：使用 write( ) 输出多行数据的实例。

```
1  # ch14_29.py
2  fn = 'out14_29.txt'
3  str1 = 'I love Python.'
4  str2 = 'Learn Python from the best book.'
5
6  with open(fn, 'w') as file_Obj:
7      file_Obj.write(str1)
8      file_Obj.write(str2)
```

执行结果 这个程序执行时在 Python Shell 窗口中看不到结果，必须到 ch14 工作目录下查看所建的 out14_29.txt 文件，同时打开可以得到下列结果。

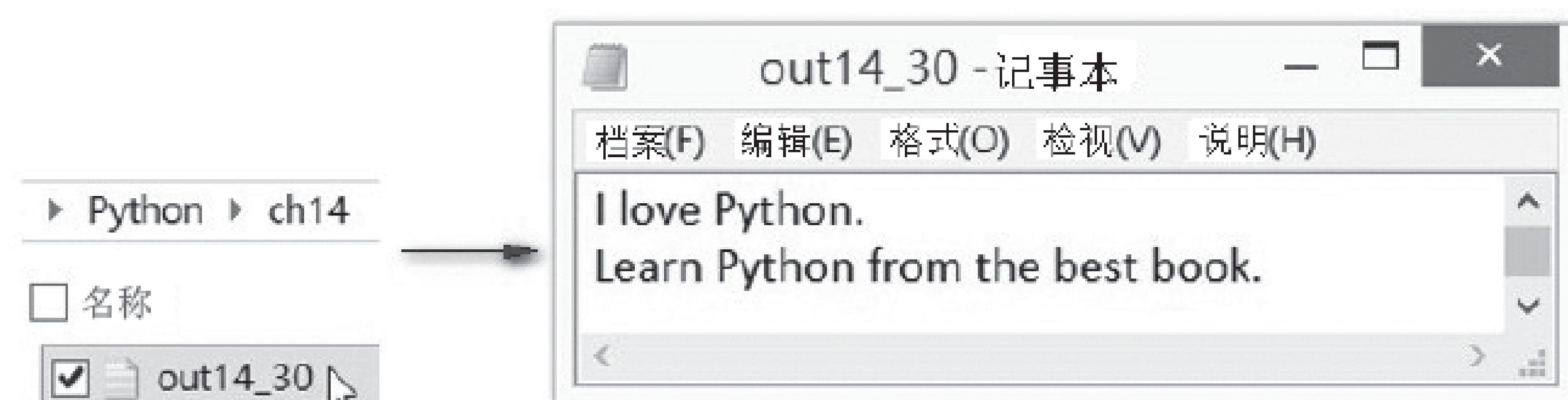


其实输出至文件时可以使用空格或换行符号，以便获得想要的输出结果。

程序实例 ch14_30.py：增加换行符号方式重新设计 ch14_29.py。

```
1  # ch14_30.py
2  fn = 'out14_30.txt'
3  str1 = 'I love Python.'
4  str2 = 'Learn Python from the best book.'
5
6  with open(fn, 'w') as file_Obj:
7      file_Obj.write(str1 + '\n')
8      file_Obj.write(str2 + '\n')
```

执行结果 这个程序执行时在 Python Shell 窗口中看不到结果，必须到 ch14 工作目录下查看所建的 out14_30.txt 文件，同时打开可以得到下列结果。



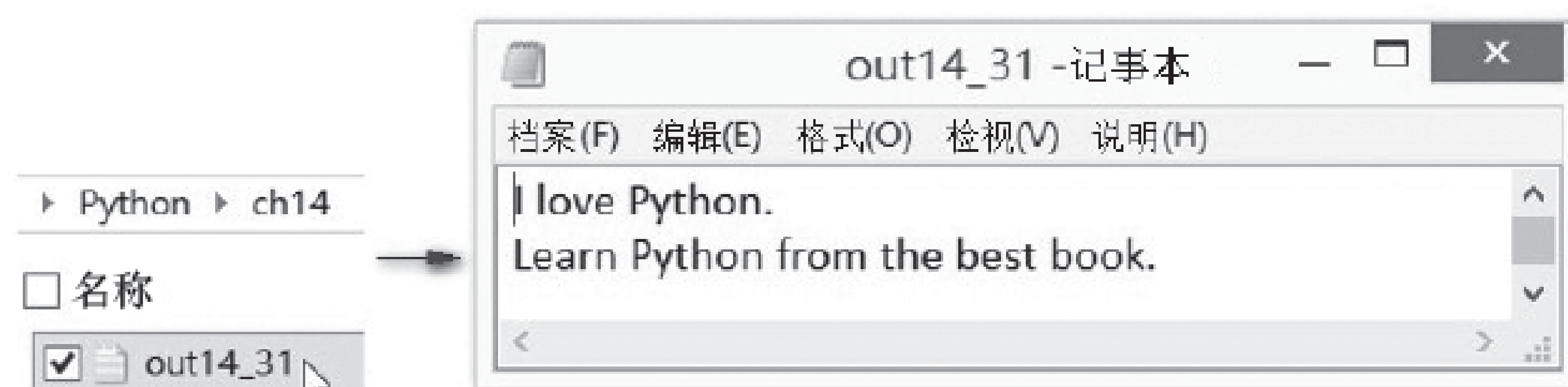
14-3-4 建立附加文件

建立附加文件主要是可以将文件输出到所打开的文件末端，当以 open( ) 打开时，需增加参数 mode='a' 或是用“a”（其实 a 是 append 的缩写）。如果用 open( ) 打开文件使用“a”参数时，若是所打开的文件不存在，Python 会打开空的文件供写入，如果所打开的文件存在，Python 在执行写入时不会清空原先的文件内容，而是将所写数据附加在原文件末端。

程序实例 ch14_31.py：建立附加文件的应用。

```
1  # ch14_31.py
2  fn = 'out14_31.txt'
3  str1 = 'I love Python.'
4  str2 = 'Learn Python from the best book.'
5
6  with open(fn, 'a') as file_Obj:
7      file_Obj.write(str1 + '\n')
8      file_Obj.write(str2 + '\n')
```

执行结果 本书 ch14 工作目录下没有 out14_31.txt 文件，所以执行第一次时，可以建立 out14_31.txt 文件，然后得到下列结果。



执行第二次时可以得到下列结果。



上述语句只要持续执行，输出数据将持续累积。

14-3-5　文件很长时的分段写入

有时候文件或字符串很长时，也可以用分批写入方式处理。

程序实例 ch14_31_1.py：将一个字符串用每次 100 字符方式写入文件，这个程序也会记录每次写入的字符数，第 2 ～ 11 行的文字取自 1-11 节 Python 之禅的内容。

```
1   # ch14_31_1.py
2   zenofPython = '''Beautiful is better than ugly.
3   Explicit is better than implicits.
4   Simple is better than complex.
5   Flat is better than nested.
6   Sparse is better than desse.
7   Readability counts.
8   Special cases aren't special enough to break the rules.
9   ...
10  ...
11  By Tim Peters'''
12
13  fn = 'out14_31_1.txt'
14  size = len(zenofPython)
15  offset = 0
16  chunk = 100
17  with open(fn, 'w') as file_Obj:
18      while True:
19          if offset > size:
20              break
21          print(file_Obj.write(zenofPython[offset:offset+chunk]))
22          offset += chunk
```

执行结果

```
==================== RESTART: D:/Python/ch14/ch14_31_1.py ====================
100
100
52
```

上述语句执行后文件夹中将有 out14_31_1.txt 文件，此文件内容如下。

```
out14_31_1 - 记事本
档案(F) 编辑(E) 格式(O) 检视(V) 说明(H)
Beautiful is better than ugly.
Explicit is better than implicits.
Simple is better than complex.
Flat is better than nested.
Sparse is better than desse.
Readability counts.
Special cases aren't special enough to break the rules.
...
...
By Tim Peters
```

从上述执行结果可以看到，写了 3 次，第 3 次是 52 个字符。

14-4 读取和写入二进制文件

14-4-1 复制二进制文件

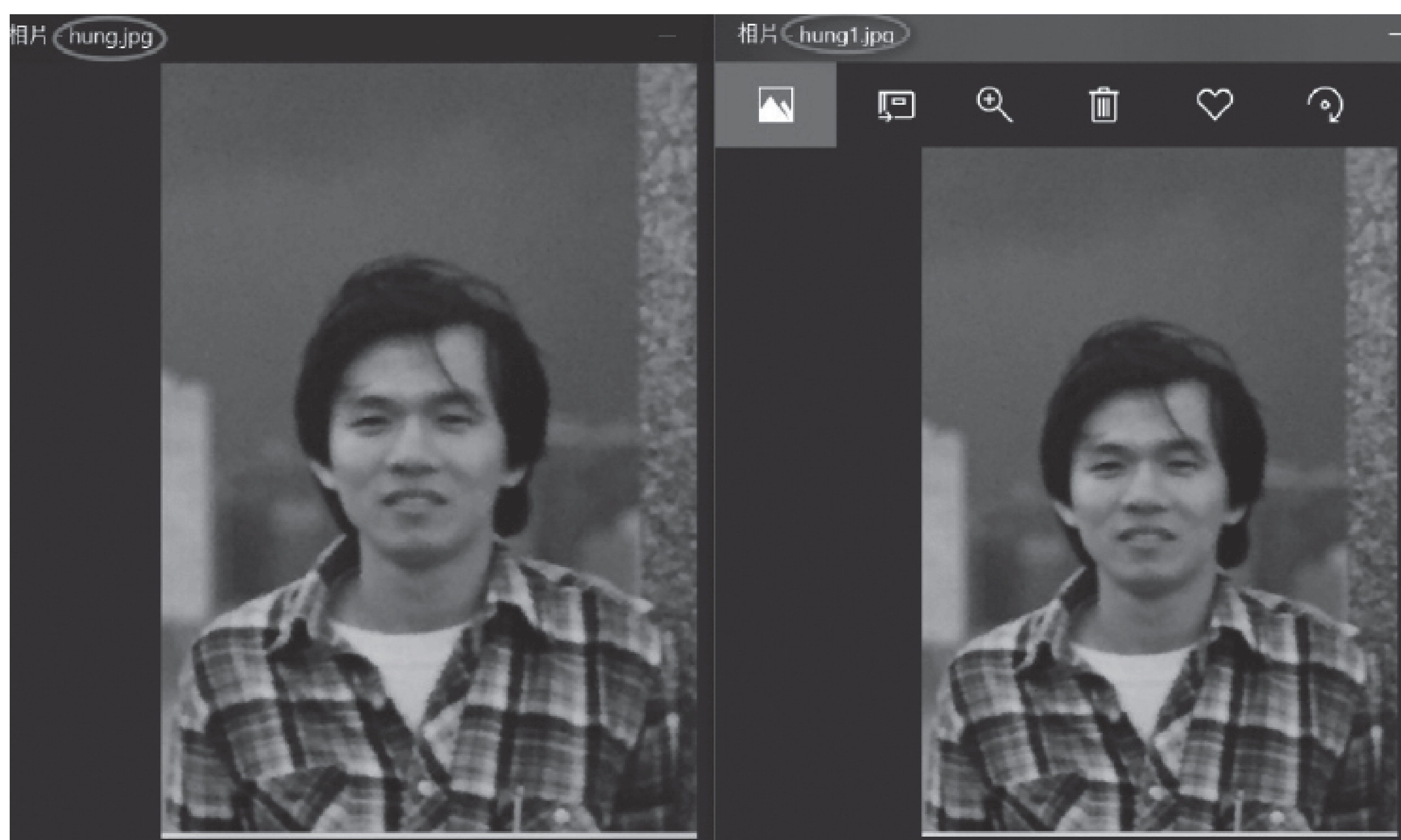
一般图文件、语音文件等都是二进制文件，如果要打开二进制文件，在使用 open( ) 时需要使用‘rb’，要写入二进制文件，在使用 open( ) 时需要使用‘wb’。

程序实例 ch14_31_2.py：复制图片文件，图片是二进制文件，这个程序会复制 hung.jpg，新复制的文件是 nhung.jpg。

```
1  # ch14_31_2.py
2  src = 'hung.jpg'
3  dst = 'hung1.jpg'
4  tmp = ''
5
6  with open(src, 'rb') as file_rd:
7      tmp = file_rd.read()
8      with open(dst, 'wb') as file_wr:
9          file_wr.write(tmp)
```

执行结果 本 Python Shell 窗口中不会有任何执行结果，不过可以在 ch14 文件夹下看到 hung.jpg 和 hung1.jpg（这是新复制的文件）。





14-4-2　随机读取二进制文件

在使用 Python 读取二进制文件时，可以随机控制读写指针的位置，也就是可以不必从头开始读取，读了每个字节后才可以读到文件最后位置。整个过程是使用 tell( ) 和 seek( ) 方法，tell( ) 可以返回从文件开头算起，目前读写指针的位置，以字节为单位。seek( ) 方法可以让目前读写指针跳到指定位置，语法如下。

```
offsetValue = seek(offset, origin)
```

seek( ) 方法会返回目前读写指针相对整体文件的位移值。其中，origrin 的意义如下。

origin 是 0（预设），读写指针移至开头算起的第 offset 个字节位置。

origin 是 1，读写指针移至目前位置算起的第 offset 个字节位置。

origin 是 2，读写指针移至相对结尾的第 offset 个字节位置。

程序实例 ch14_31_3.py：建立一个 0 ～ 255 的二进制文件。

```
1  # ch14_31_3.py
2  dst = 'bdata'
3  bytedata = bytes(range(0,256))
4  with open(dst, 'wb') as file_dst:
5      file_dst.write(bytedata)
```

执行结果　这只是建立一个 bdata 二进制文件。

程序实例 ch14_31_4.py：随机读取二进制文件的应用。

```
1  # ch14_31_4.py
2  src = 'bdata'
3
4  with open(src, 'rb') as file_src:
5      print("目前位移 : ", file_src.tell())
6      file_src.seek(10)
7      print("目前位移 : ", file_src.tell())
8      data = file_src.read()
9      print("目前內容 : ", data[0])
10     file_src.seek(255)
11     print("目前位移 : ", file_src.tell())
12     data = file_src.read()
13     print("目前內容 : ", data[0])
```

执行结果

```
==================== RESTART: D:/Python/ch14/ch14_31_4.py ====================
目前位移 :  0
目前位移 :  10
目前内容 :  10
目前位移 :  255
目前内容 :  255
```

14-5 shutil 模块

这个模块有提供一些方法让我们可以在 Python 程序内执行文件或目录的复制、删除、更改位置和更改名称。当然在使用前需加上下列加载模块指令。

```
import  shutil            # 加载模块指令
```

14-5-1 文件的复制 copy( )

在 shutil 模块可以使用 copy( ) 执行文件的复制，语法格式如下。

```
shutil.copy(source, destination)
```

上述语句可将 source 文件复制到 destination 目的位置，执行前 source 文件一定要存在，否则会产生错误。另外，这个方法也可以复制二进制文件。

程序实例 ch14_32.py：执行文件复制的应用。

```
1  # ch14_32.py
2  import shutil
3
4  shutil.copy('source.txt', 'dest.txt')        # 目前工作目录文件复制
5  shutil.copy('source.txt', 'D:\\Python')      # 目前工作目录文件复制至D:\Python
6  shutil.copy('D:\\Python\\source.txt', 'D:\\dest.txt') # 不同工作目录文件复制
```

执行结果 这个程序没有列出任何数据，说明如下。

第 4 行，目前工作目录 source.txt 复制一份放在目前工作目录下，文件名是 dest.txt。

第 5 行，目前工作目录 source.txt 使用相同名称复制一份放在 D:\Python。

第 6 行，D:\Python 目录下 source.txt 复制一份放在 D:\ 下，名称是 dest.txt。

14-5-2 目录的复制 copytree( )

copytree( ) 的语法格式与 copy( ) 相同，只不过是复制目录，复制时目录底下的子目录或文件也将被复制。此外，执行前来源目录一定要存在，否则会产生错误。

程序实例 ch14_33.py：目录复制的应用。

```
1  # ch14_33.py
2  import shutil
3
4  shutil.copytree('old14', 'new14')                  # 目前工作目录的目录复制
5  shutil.copytree('D:\\Python\\old14', 'D:\\new14')  # 不同工作目录的目录复制
```

执行结果 这个程序没有列出任何数据，它的说明如下：

第 4 行，目前工作目录 old14 复制一份到目前工作目录，名称是 new14。

第 5 行，D:\Python 复制 old14 目录至 D:\，名称是 new14。

14-5-3　文件的移动 move( )

在 shutil 模块中可以使用 move( ) 执行文件的移动，语法格式如下。

```
shutil.move(source, destination)
```

上述语句可将 source 文件移动到 destination 目的位置，执行前 source 文件一定要存在，否则会产生错误，执行后 source 文件将不再存在。

程序实例 ch14_34.py：将目前目录的 data34.txt 移至目前目录的 test34 子目录。

```
1  # ch14_34.py
2  import shutil
3
4  shutil.move('data34.txt', '.\\test34')  # 移动目前工作目录data34.txt
```

执行结果 执行前目前目录底下需有 test34 子目录，然后可以得到下列结果。



14-5-4　文件名的更改 move( )

在移动过程中如果 destination 路径有含文件名，则可以达到更改文件名称的效果。

程序实例 ch14_35.py：在同目录下更改文件名。

```
1  # ch14_35.py
2  import shutil
3
4  shutil.move('data35.txt', 'out35.txt')  # 更改文件名
```

执行结果 上述程序会将 data35.txt 改名为 out35.txt。

在文件移动过程中若是 destination 的目录不存在，也将造成文件名的更改。

程序实例 ch14_36.py：文件名更改的另一种状况。

```
1  # ch14_36.py
2  import shutil
3
4  shutil.move('data36.txt', 'D:\\Python\\out36')  # out36不存在
```

执行结果 下面是验证结果。



上述语句执行前 D:\Python\out36.txt 不存在，将造成以 D:\Python\out36.txt 存储此文件。

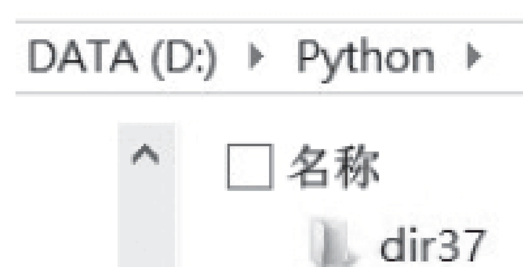
14-5-5 目录的移动 move( )

这个 move( ) 也可以执行目录的移动，在移动时子目录也将随着移动。

程序实例 ch14_37.py：将目前工作目录的子目录 dir37 移至 D:\Python 目录下。

```
1  # ch14_37.py
2  import shutil
3
4  shutil.move('dir37', 'D:\\Python')
```

执行结果 下面是验证结果。



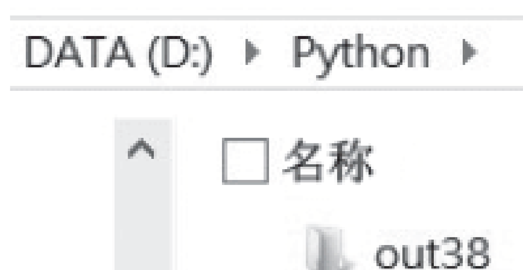
14-5-6 更改目录名称 move( )

如果在移动过程中 destination 的目录不存在，此时就可以达到更改目录名称的目的了，此时甚至路径名称也可能更改。

程序实例 ch14_38.py：将目前子目录 dir38 移动至 D:\Python，同时改名为 out38。

```
1  # ch14_38.py
2  import shutil
3
4  shutil.move('dir38', 'D:\\Python\\out38')
```

执行结果



14-5-7 删除有数据的目录 rmtree( )

os 模块的 rmdir( ) 只能删除空的目录，如果要删除含有数据文件的目录，需使用本节所介绍的 rmtree( )。

程序实例 ch14_39.py：删除 dir39 目录，这个目录下有数据文件 data39.txt。

```
1  # ch14_39.py
2  import shutil
3
4  shutil.rmtree('dir39')
```

执行结果 执行后下列 D:\Python\ch14\dir39 将被删除。



14-5-8　安全删除文件或目录 send2trash( )

Python 内建的 shutil 模块在删除文件后就无法复原了，目前有一个第三方的模块 send2trash，执行删除文件或文件夹后是将被删除的文件放在回收站，如果后悔可以撤回。不过在使用此模块前需先下载这个外部模块。可以进入安装 Python 的文件夹，然后在 DOS 环境安装此模块，安装指令如下。

```
pip install send2trash
```

有关安装第 3 方模块的方法可参考附录 B，安装完成后就可以使用下列方式删除文件或目录了。

```
import    send2trash                         # 导入 send2trash 模块
send2trash.send2trash（文件或文件夹）          # 语法格式
```

程序实例 ch14_40.py：删除文件 data40.txt，未来可以在回收站找到此文件。

```
1  # ch14_40.py
2  import send2trash
3
4  send2trash.send2trash('data40.txt')
```

执行结果　下列是回收站中找到此 data40.txt 的结果。



14-6　文件压缩与解压缩

Windows 操作系统有提供功能将一般文件或目录执行压缩，压缩后的扩展名是 zip。Python 内有 zipFile 模块也可以将文件或目录执行压缩以及解压缩。当然程序开头需要加上下列指令导入此模块。

```
import   zipFile
```

14-6-1　执行文件或目录的压缩

执行文件压缩前首先要使用 ZipFile( ) 方法建立一份压缩后的文件名，在这个方法中另外要加上 ‘w’ 参数，注明未来是供 write( ) 方法写入。

```
fileZip = zipfile.ZipFile('out.zip', 'w')    # out.zip 是未来存储压缩结果
```

上述 fileZip 和 out.zip 都可以自由设置名称。fileZip 是**压缩文件对象**，代表的是 out.zip，未来将被压缩的文件数据写入**此对象**，就可以执行将结果存入 out.zip。由于 ZipFile( ) 无法执行整个目录的压缩，不过可用循环方式将目录底下的文件压缩，即可达到压缩整个目录的目的。

程序实例 ch14_41.py：这个程序会将目前工作目录底下的 zipdir41 目录压缩，压缩结果存储在 out41.zip 内。这个程序执行前的 zipdir41 内容如下。

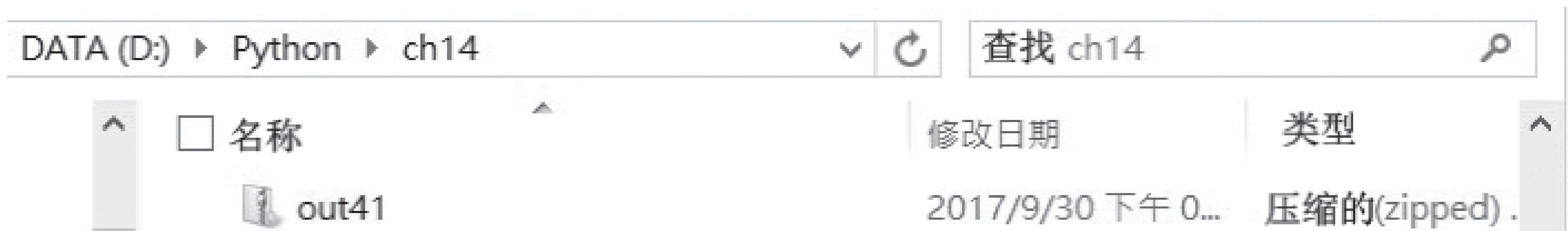


下面是程序内容。

```
1  # ch14_41.py
2  import zipfile
3  import glob, os
4
5  fileZip = zipfile.ZipFile('out41.zip', 'w')
6  for name in glob.glob('zipdir41/*'):        # 遍历zipdir41目录
7      fileZip.write(name, os.path.basename(name), zipfile.ZIP_DEFLATED)
8
9  fileZip.close()
```

说明压缩方式

执行结果 可以在相同目录下得到下列压缩文件 out41。



14-6-2 读取 zip 文件

ZipFile 对象有 namelist( ) 方法可以返回 zip 文件内所有被压缩的文件或目录名称，同时以列表方式返回此对象。这个返回的对象可以使用 infolist( ) 方法返回各元素的属性，文件名 filename、文件大小 file_size、压缩结果大小 compress_size、文件时间 data_time。

程序实例 ch14_42.py：将 ch14_41.py 所建的 zip 文件解析，列出所有被压缩的文件，以及文件名、文件大小和压缩结果大小。

```
1  # ch14_42.py
2  import zipfile
3
4  listZipInfo = zipfile.ZipFile('out41.zip', 'r')
5  print(listZipInfo.namelist())          # 以列表列出所有压缩文件
6  print("\n")
7  for info in listZipInfo.infolist():
8      print(info.filename, info.file_size, info.compress_size)
```

执行结果

```
===================== RESTART: D:\Python\ch14\ch14_42.py =====================
['20161024洪錦魁.jpg', 'antarctica2.jpg', 'forZipTest.docx', 'IMG_1658.jpg', 'IMG_8036.jpg', 'IMG_8096.jpg', 'IMG_8957.JPG']

20161024洪錦魁.jpg 166763 166531
antarctica2.jpg 1440258 1430105
forZipTest.docx 1266045 1252488
IMG_1658.jpg 1478242 1475740
IMG_8036.jpg 2885322 2877251
IMG_8096.jpg 1473764 1471145
IMG_8957.JPG 129424 126337
>>>
```

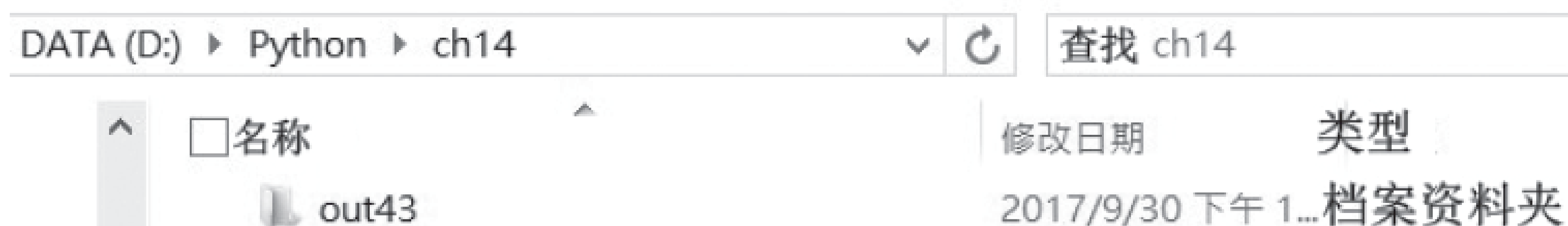
14-6-3　解压缩 zip 文件

解压缩 zip 文件可以使用 extractall( ) 方法。

程序实例 ch14_43.py：将程序实例 ch14_41.py 所建的 out41.zip 解压缩，同时将压缩结果存入 out43 目录。

```
1   # ch14_43.py
2   import zipfile
3
4   fileUnZip = zipfile.ZipFile('out41.zip')
5   fileUnZip.extractall('out43')
6   fileUnZip.close()
```

执行结果



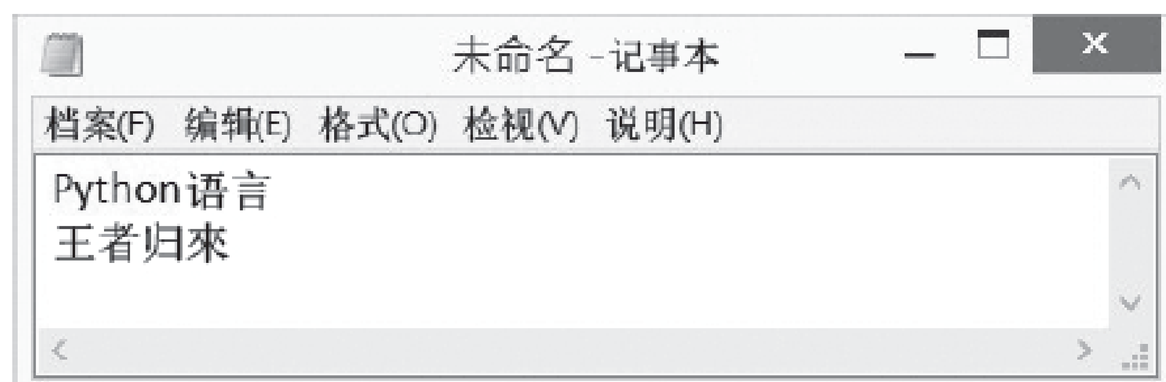
14-7　认识编码格式 encode

目前为止所谈到的文本文件（.txt）的打开的编码都是使用 Windows 操作系统的默认方式，文本模式下常用的编码方式有 utf-8 和 cp950。使用 open( ) 打开文件时，可以增加另一个常用的参数 encoding，open( ) 的语法将如下所示。

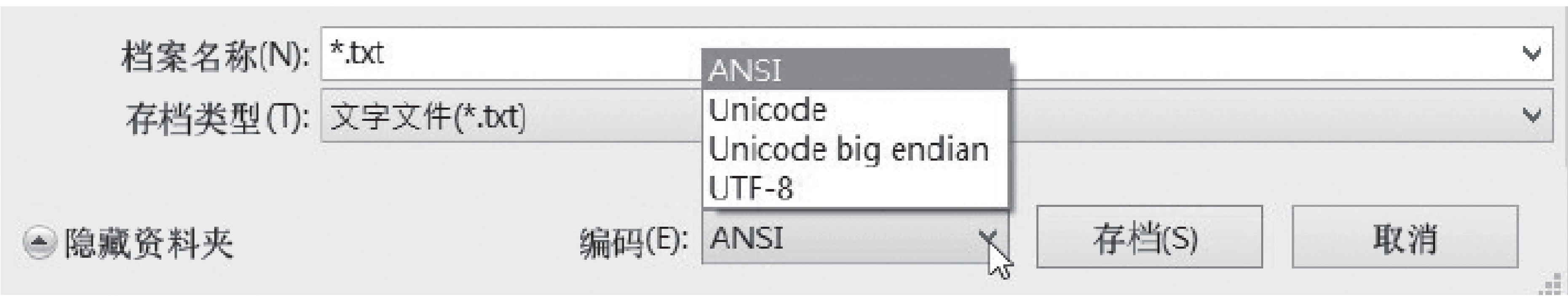
```
file_Obj = open(file, mode="r", encoding="cp950")
```

14-7-1　繁体中文 Windows 操作系统记事本默认的编码

打开中文 Windows 操作系统的记事本建立下列文件。



执行“文件”→“另存为”指令。

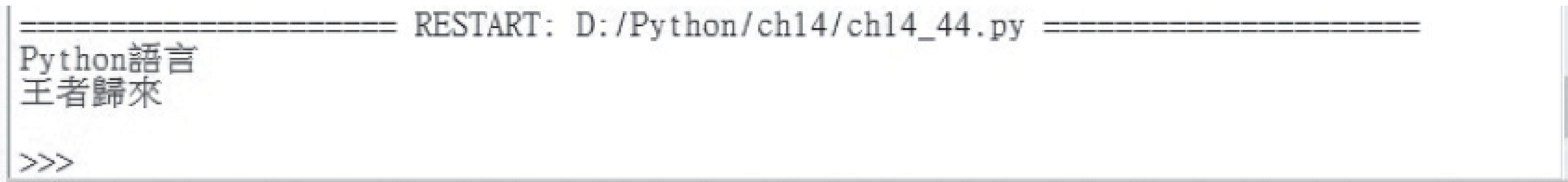


在繁体中文 Windows 环境下，上述默认编码是 ANSI，在这个编码格式下，在 Python 的 open( ) 内可以使用预设的 encoding="cp950" 编码，因为这是 Python 预设的所以可以省略此参数。请将上述文件使用默认的 ANSI 编码存至 ansi14_44.txt。

程序实例 ch14_44.py：使用 encoding="950" 打开 ansi14_44.txt，然后输出。

```
1  # ch14_44.py
2
3  fn = 'ansi14_44.txt'                    # 设置要打开的文件
4  file_Obj =  open(fn, encoding='cp950')  # 用预设encoding='cp950'打开文件
5  data = file_Obj.read()                  # 读取文件到变量data
6  file_Obj.close()                        # 关闭文件对象
7  print(data)                             # 输出变量data相当于输出文件
```

执行结果


```
==================== RESTART: D:/Python/ch14/ch14_44.py ====================
Python語言
王者歸來

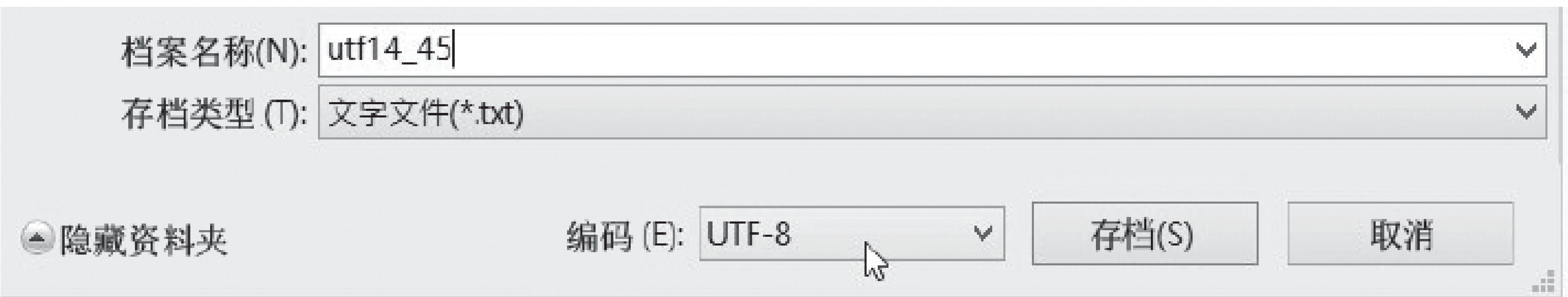
>>>
```

14-7-2 utf-8 编码

utf-8 英文全名是 8-bit Unicode Transformation Format，这是一种适合多语系的编码规则，主要是使用可变长度字节方式存储字符，以节省内存空间。例如，对于英文字母而言是使用 1 字节空间存储即可，对于含有附加符号的希腊文、拉丁文或阿拉伯文等则用两个字节空间存储字符，中文字则是以 3 字节空间存储字符，只有极少数的平面辅助文字需要 4 字节空间存储字符。也就是说，这种编码规则已经包含全球所有语言的字符了，所以采用这种编码方式设计网页时，其他国家的浏览器只要支持 utf-8 编码都可显示。例如，美国人即使使用英文版的 Internet Explorer 浏览器，也可以正常显示中文。

另外，有时我们在网络世界浏览其他国家的网页时，会发生显示乱码情况，主要原因就是对方网页设计师并没有将此属性设为 utf-8。例如，早期最常见的是，简体中文的编码是 gb2312，这种编码方式是以 2 字符组存储一个简体中文字，由于这种编码方式不适用多语系，无法在繁体中文 Windows 环境中使用，如果网页设计师采用此编码，将造成中国香港、中国澳门或中国台湾地区繁体中文 Widnows 的用户在繁体中文窗口环境浏览此网页时出现乱码。

其实 utf-8 是国际通用的编码，如果使用 Linux 或 Max OS，一般也是用国际编码，所以如果打开文件发生错误，请先检查文件的编码格式。打开 ansi14_44.txt 文件，然后执行“另存为”命令，此时编码规则请选择 utf-8 编码，将文件存入 utf14_45.txt，如下所示。



程序实例 ch14_45.py：重新设计 ch14_44.py，使用 encoding='950' 打开文件发生错误的实例。

```
1  # ch14_45.py
2
3  fn = 'utf14_45.txt'                          # 设置要打开的文件
4  file_Obj =  open(fn, encoding='cp950')       # 用预设encoding='cp950'打开文件
5  data = file_Obj.read()                       # 读取文件到变量data
6  file_Obj.close()                             # 关闭文件对象
7  print(data)                                  # 输出变量data相当于输出文件
```

执行结果

```
==================== RESTART: D:\Python\ch14\ch14_45.py ====================
Traceback (most recent call last):
  File "D:\Python\ch14\ch14_45.py", line 5, in <module>
    data = file_Obj.read()                # 读取档案到变量data
UnicodeDecodeError: 'cp950' codec can't decode byte 0x9e in position 11: illegal
 multibyte sequence
>>>
```

上述结果很明显指出是译码 decode 错误。

程序实例 ch14_46.py：重新设计 ch14_45.py，使用 encoding='utf-8'。

```
1  # ch14_46.py
2
3  fn = 'utf14_45.txt'                          # 设置要打开的文件
4  file_Obj =  open(fn, encoding='utf-8')       # 用encoding='utf-8'打开文件
5  data = file_Obj.read()                       # 读取文件到变量data
6  file_Obj.close()                             # 关闭文件对象
7  print(data)                                  # 输出变量data相当于输出文件
```

执行结果

```
==================== RESTART: D:\Python\ch14\ch14_46.py ====================
Python语言
王者归来

>>>
```

14-7-3　认识 utf-8 编码的 BOM

使用中文 Windows 操作系统的记事本以 utf-8 编码时，操作系统会在文件前端增加**字节顺序记号**（Byte Order Mark, BOM），俗称**文件前端代码**，主要功能是判断文字以 Unicode 表示时，字节的排序方式。

程序实例 ch14_47.py：重新设计 ch14_20.py，使用逐行读取方式读取 utf-8 编码格式的 utf14_45.txt 文件，验证 BOM 的存在。

```
1  # ch14_47.py
2
3  fn = 'utf14_45.txt'                                  # 设置要打开的文件
4  with open(fn, encoding='utf-8') as file_Obj:         # 打开utf-8文件
5      obj_list = file_Obj.readlines()                  # 每次读一行
6
7  print(obj_list)                                      # 打印串行
```

执行结果

```
==================== RESTART: D:\Python\ch14\ch14_47.py ====================
['\ufeffPython语言\n', '王者归来\n']
>>>
```

从上述执行结果可以看到 \ufeff 字符，其实 u 代表这是 Unicode 编码格式，fe 和 ff 是十六进制的编码格式，代表编码格式。在 utf-8 的编码中有两种编码格式主张，有一派主张数值较大的字节要放在前面，这种方式称为 Big Endian（BE）系统。另一派主张数值较小的字节要放在前面，这种方式称为 Little Endian（LE）系统。目前，Windows 系统的编法是 LE 系统，它的 BOM 内容是 \ufeff，由于目前没有所谓的 \ufffe 内容，所以一般就用 BOM 内容是 \ufeff 代表这是 LE 的编码系统。这两个字符在 Unicode 中不占空间，所以许多时候感觉不到它们的存在。

使用 open( ) 函数时，也可以很明确地使用 encoding='utf-8-sig'，这时即使是逐行读取也可以将 BOM 去除。

程序实例 ch14_48.py：重新设计 ch14_47.py，使用 encoding='utf-8-sig' 格式。

```
1  # ch14_48.py
2
3  fn = 'utf14_45.txt'                              # 设置要打开的文件
4  with open(fn, encoding='utf-8-sig') as file_Obj:  # 打开utf-8文件
5      obj_list = file_Obj.readlines()              # 每次读一行
6
7  print(obj_list)                                  # 打印列表
```

执行结果 从执行结果可以看到 \ufeff 字符没有了。

```
==================== RESTART: D:\Python\ch14\ch14_48.py ====================
['Python语言\n', '王者归来\n']
>>>
```

另外有一些专业的文字编辑软件提供使用 utf-8 格式但是去除 BOM 的方式存盘，例如，NotePad 软件，操作时可在菜单栏的“编码”一项中选择“编译成 utf-8 码（文件首无 Bom）”，最后将执行结果存入 utf14_49.txt。

程序实例 ch14_49.py：重新设计 ch14_47.jpy，这次改读取 utf14_49.txt，并观察执行结果，看不到 \ufeff 字符了。

```
1  # ch14_49.py
2
3  fn = 'utf14_49.txt'                          # 设置要打开的文件
4  with open(fn, encoding='utf-8') as file_Obj:  # 打开utf-8文件
5      obj_list = file_Obj.readlines()          # 每次读一行
6
7  print(obj_list)                              # 打印列表
```

执行结果

```
==================== RESTART: D:\Python\ch14\ch14_49.py ====================
['Python语言\n', '王者归来\n']
>>>
```

14-8 剪贴板的应用

剪贴板的功能是属于第三方 pyperclip 模块内，使用前需使用下列方式安装此模块，更多知识可参考附录 B。

```
pip install pyperclip
```

然后在程序前面加上下列导入 pyperclip 模块功能。

```
import pyperclip
```

安装完成后就可以使用下面两个方法。

（1）copy( )：可将列表数据复制至剪贴板。

（2）paste( )：将剪贴板数据复制回字符串变量。

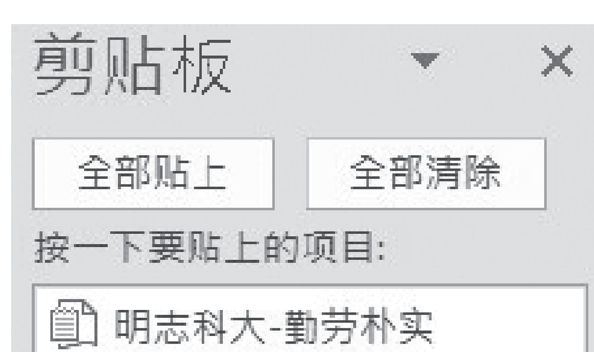
程序实例 ch14_50.py：将数据复制至剪贴板，再将剪贴板数据复制回字符串变量 string，同时打印 string 字符串变量。

```
1  # ch14_50.py
2  import pyperclip
3
4  pyperclip.copy('明志科大-勤劳朴实')        # 将字符串复制至剪贴板
5  string = pyperclip.paste()               # 从剪贴板复制回string
6  print(string)                            # 打印
```

执行结果

```
==================== RESTART: D:\Python\ch14\ch14_50.py ====================
明志科大-勤劳朴实
>>>
```

其实上述执行第 4 行后，如果打开剪贴板（可打开 Word 再进入剪贴板功能）可以看到“明志科大 - 勤劳朴实”字符串已经出现在剪贴板中。程序第 5 行则是将剪贴板数据复制至 string 字符串变量，第 6 行则是打印 string 字符串变量。

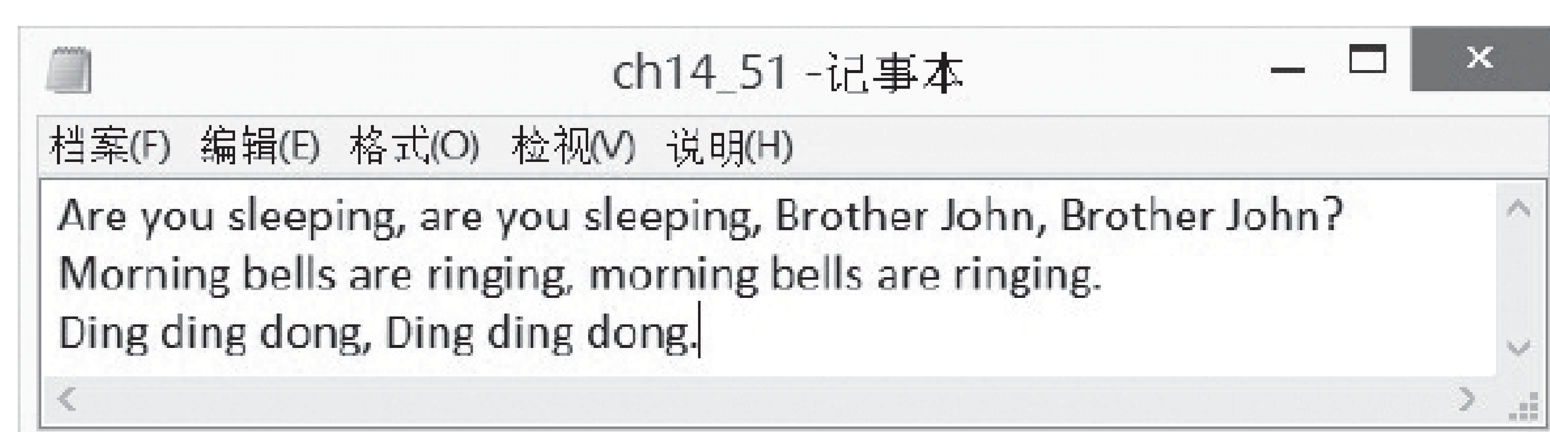


14-9　专题——分析文件 / 加密文件

14-9-1　以读取文件方式处理分析文件

我们有学过字符串、列表、字典、设计函数、文件打开与读取文件，本节将举一个实例可以应用上述概念。

程序实例 ch14_51.py：有一首两只老虎的儿歌放在 ch14_51.txt 文件内，其实这首耳熟能详的儿歌是法国歌曲，原歌词如下：



这个程序主要是列出每个单词出现的次数，为了简单，全部单词改成小写显示。这个程序将用字典保存执行结果，字典的键是单词，字典的值是单词出现次数。为了让读者了解本程序的每个步骤，笔者将输出每一个阶段的变化。

```
1  # ch14_51.py
2  def modifySong(songStr):              # 将歌曲的标点符号用空字符取代
3      for ch in songStr:
4          if ch in ".,?":
5              songStr = songStr.replace(ch,'')
6      return songStr                    # 返回取代结果
7
8  def wordCount(songCount):
9      global mydict
10     songList = songCount.split()      # 将歌曲字符串转成列表
11     print("以下是歌曲列表")
12     print(songList)
13     mydict = {wd:songList.count(wd) for wd in set(songList)}
14
15 fn = "ch14_51.txt"
16 with open(fn) as file_Obj:            # 打开歌曲文件
17     data = file_Obj.read()            # 读取歌曲文件
18     print("以下是所读取的歌曲")
19     print(data)                       # 打印歌曲文件
20
21 mydict = {}                           # 空字典未来储存单字计数结果
22 print("以下是将歌曲大写字母全部改成小写同时将标点符号用空字符取代")
23 song = modifySong(data.lower())
24 print(song)
25
26 wordCount(song)                       # 执行歌曲单词计数
27 print("以下是最后执行结果")
28 print(mydict)
```

执行结果

```
===================== RESTART: D:\Python\ch14\ch14_51.py =====================
以下是所读取的歌曲
Are you sleeping, are you sleeping, Brother John, Brother John?
Morning bells are ringing, morning bells are ringing.
Ding ding dong, Ding ding dong.
以下是将歌曲大写字母全部改成小写同时将标点符号用空字符取代
are you sleeping are you sleeping brother john brother john
morning bells are ringing morning bells are ringing
ding ding dong ding ding dong
以下是歌曲列表
['are', 'you', 'sleeping', 'are', 'you', 'sleeping', 'brother', 'john', 'brother
', 'john', 'morning', 'bells', 'are', 'ringing', 'morning', 'bells', 'are', 'rin
ging', 'ding', 'ding', 'dong', 'ding', 'ding', 'dong']
以下是最后执行结果
{'are': 4, 'john': 2, 'ding': 4, 'ringing': 2, 'sleeping': 2, 'dong': 2, 'brothe
r': 2, 'morning': 2, 'bells': 2, 'you': 2}
```

14-9-2 加密文件

13-10-4 已经介绍了加密文件的概念，但是那只是为一个字符串执行加密，更进一步地我们可以设计为一个文件加密，一般文件中有‘\n’或‘\t’字符，所以必须在加密与解密字典内增加考虑这两个字符。

程序实例 ch14_52.py：这个程序将加密由 Tim Peters 所写的“Python 之禅”。首先将此“Python 之禅”建立在 ch14 文件夹内，文件名是 zenofPython.txt，然后读取此文件，最后列出加密结果。读者需留意第 11 行，不可打印字符只删除最后 3 个字符。

```
1  # ch14_52.py
2  import string
3
4  def encrypt(text, encryDict):                # 加密文件
5      cipher = []
6      for i in text:                           # 执行每个字符加密
7          v = encryDict[i]                     # 加密
8          cipher.append(v)                     # 加密结果
9      return ''.join(cipher)                   # 将列表转成字符串
10
11 abc = string.printable[:-3]                  # 取消不可打印字符
12 subText = abc[-3:] + abc[:-3]                # 加密字符串字符串
13 encry_dict = dict(zip(subText, abc))         # 建立字典
14
15 fn = "zenofPython.txt"
16 with open(fn) as file_Obj:                   # 打开文件
17     msg = file_Obj.read()                    # 读取文件
18
19 ciphertext = encrypt(msg, encry_dict)
20
21 print("原始字符串")
22 print(msg)
23 print("加密字符串")
24 print(ciphertext)
```

执行结果

```
===================== RESTART: D:\Python\ch14\ch14_52.py =====================
原始字符串
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
加密字符串
Wkh0#hq0ri0SBwkrq/0eB0Wlp0Shwhuv22Ehdxwlixo0lv0ehwwhu0wkdq0xjoB;2HAsolflw0lv0ehw
whu0wkdq0lpsolflw;2Vlpsoh0lv0ehwwhu0wkdq0frpsohA;2FrpsohA0lv0ehwwhu0wkdq0frpsolf
dwhg;2Iodw0lv0ehwwhu0wkdq0qhvwhg;2Vsduvh0lv0ehwwhu0wkdq0ghqvh;2UhdgdelolwB0frxqw
v;2Vshfldo0fdvhv0duhq*w0vshfldo0hqrxjk0wr0euhdn0wkh0uxohv;2Dowkrxjk0sudfwlfdolwB
0ehdwv0sxulwB;2Huuruv0vkrxog0qhyhu0sdvv0vlohqwoB;2Xqohvv0hAsolflwoB0vlohqfhg;2Lq
0wkh0idfh0ri0dpeljxlwB/0uhixvh0wkh0whpswdwlrq0wr0jxhvv;2Wkhuh0vkrxog0eh0rqh::0dq
g0suhihudeoB0rqoB0rqh0::reylrxv0zdB0wr0gr0lw;2Dowkrxjk0wkdw0zdB0pdB0qrw0eh0reylr
xv0dw0iluvw0xqohvv0Brx*uh0Gxwfk;2Qrz0lv0ehwwhu0wkdq0qhyhu;2Dowkrxjk0qhyhu0lv0riw
hq0ehwwhu0wkdq0-uljkw-0qrz;2Li0wkh0lpsohphqwdwlrq0lv0kdug0wr0hAsodlq/0lw*v0d0edg
0lghd;2Li0wkh0lpsohphqwdwlrq0lv0hdvB0wr0hAsodlq/0lw0pdB0eh0d0jrrg0lghd;2Qdphvsdf
hv0duh0rqh0krqnlqj0juhdw0lghd0::0ohw*v0gr0pruh0ri0wkrvh$
```

如何验证上述加密正确？最好的方式是为上述加密结果解密，这将是读者的习题。

习题

1. 请输入一个目录，如果此目录存在则输出“xxx 已经存在”，如果此目录不存在则先输出“目录不存在”，建立此目录，然后输出“建立此目录”。(14-1 节)

```
===================== RESTART: D:\Python\ex\ex14_1.py =====================
请输入目录 : abc
abc 已经存在
>>>
===================== RESTART: D:\Python\ex\ex14_1.py =====================
请输入目录 : abc
abc 已经存在
```

2. 请输入一个文件，如果文件不存在则输出“xxx 文件不存在”，如果文件存在则输出此文件的大小。（14-1 节）

```
===================== RESTART: D:\Python\ex\ex14_2.py =====================
请输入文件 : ex14_1.py
ex14_1.py : 236
>>>
===================== RESTART: D:\Python\ex\ex14_2.py =====================
请输入文件 : tmp.py
tmp.py 文件不存在
```

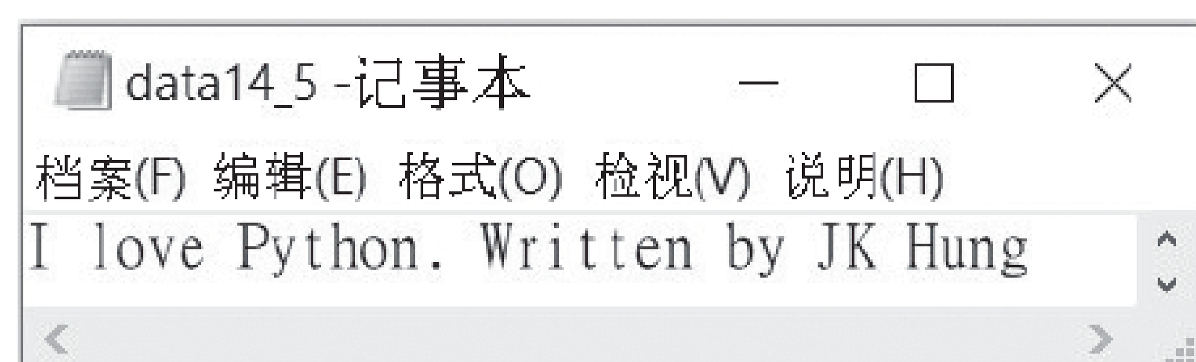
3. 请重新设计 ch14_13.py，输入特定目录，如果此目录不存在，则列出“xxx 此目录不存在”。如果此目录存在，本程序会列出特定目录的所有文件和大小，同时列出此目录所有文件数量和全部大小。（14-1 节）

```
===================== RESTART: D:\Python\ex\ex14_3.py =====================
请输入目录 : D:\\Python\\ch1
ch1_1.py : 24
ch1_2.py : 62
ch1_3.py : 135
ch1_4.py : 139
全部文件数量是 =  4
全部文件大小是 =  360
>>>
===================== RESTART: D:\Python\ex\ex14_3.py =====================
请输入目录 : aaa
aaa 目录不存在
```

4. 请更改设计 ch14_7.py，让各行字符串在同一行输出，下列是执行结果。（14-2 节）

```
========= RESTART: D:\Python\ex\ex14_4.py ===========
itute of TechnologyMing-Chi University of TechnologyI
```

5. 本章讲解了读取文件的知识，也讲解了写入文件的知识，请设计一个 copy 程序，将一个文件写入另一个文件内。程序执行时会先要求输入原始文件的文件名，然后要求输入目的文件的文件名，程序会将原始文件的内容写入目的文件内。本书 ch14 文件夹有下列测试文件 data14_5.txt。（14-3 节）



下列是执行示范输出。

```
===================== RESTART: D:\Python\ex\ex14_5.py =====================
请输入来源文件 : data14_5.txt
请输入目标文件 : out14_5.txt
```

执行完后可以在目前文件夹看到 out14_5.txt 文件，它的内容将和 data14_5.txt 相同。（14-3 节）

6. 有 5 个字符串内容如下。（14-3 节）

```
str1 = 'Python'
str2 = 'Author : Jiin-Kwei Hung'
```

```
str3 = 'DeepMind Co.'
str4 = 'DeepStone Corporation'
str5 = 'Deep Learning'
```

请依上述字符串执行下列工作。

（1）分 5 行输出，将执行结果存入 ex14_6_1.txt。

```
Python
Author : Jiin-Kwei Hung
DeepMind Co.
DeepStone Corporation
Deep Learning
```

（2）同一行输出，彼此间不空格，将执行结果存入 ex14_6_2.txt。

```
PythonAuthor : Jiin-Kwei HungDeepMind Co.DeepStone CorporationDeep Learning
```

（3）同一行输出，彼此间空两格，将执行结果存入 ex14_6_3.txt。

```
Python  Author : Jiin-Kwei Hung  DeepMind Co.  DeepStone Corporation  Deep Learning
```

7. 请一次读取 ex14_6_1.txt，然后输出到屏幕。(14-3 节)

```
===================== RESTART: D:\Python\ex\ex14_7.py =====================
Python
Author : Jiin-Kwei Hung
DeepMind Co.
DeepStone Corporation
Deep Learning
```

8. 请一次一行读取 ex14_6_1.txt，然后输出到屏幕。(14-3 节)

```
===================== RESTART: D:\Python\ex\ex14_8.py =====================
Python

Author : Jiin-Kwei Hung

DeepMind Co.

DeepStone Corporation

Deep Learning
```

9. 请一次一行读取 ex14_6_1.txt，然后处理成一行且彼此间不空格，然后输出到屏幕。(14-3 节)

```
===================== RESTART: D:\Python\ex\ex14_9.py =====================
PythonAuthor : Jiin-Kwei HungDeepMind Co.DeepStone CorporationDeep Learning
```

10. 请参考 ch14_31_2.py，设计 copy 二进制文件，以图文件为实例，其中，来源文件和目标文件必须由屏幕输入。(14-4 节)

```
===================== RESTART: D:\Python\ex\ex14_10.py =====================
请输入来源图文件: hung.jpg
请输入目的图文件: hung1.jpg
```

11. 请参考 ch14_32.py，重新设计上述 ex14_10.py，来源文件和目标文件必须由屏幕输入。(14-5 节)

```
===================== RESTART: D:\Python\ex\ex14_11.py =====================
请输入来源图文件 : hung.jpg
请输入目的图文件 : hung2.jpg
```

12. 请参考 ch14_41.py 执行特定目录内容的文件压缩，必须从屏幕输入要压缩的目录，以及存储压缩结果的文件。在习题文件夹下有 zipabc 目录，这个文件夹内容与 ch14 文件夹的 zipdir41 相同，下面是示范输出。(14-6 节)

```
====================== RESTART: D:\Python\ex\ex14_12.py ======================
请输入要压缩的目录 : zipabc
请输入保存压缩文件的名称 : zip14_12.zip
```

上述保存压缩文件的名称是 zip14_12.zip，所以可以在目前目录看到此文件。

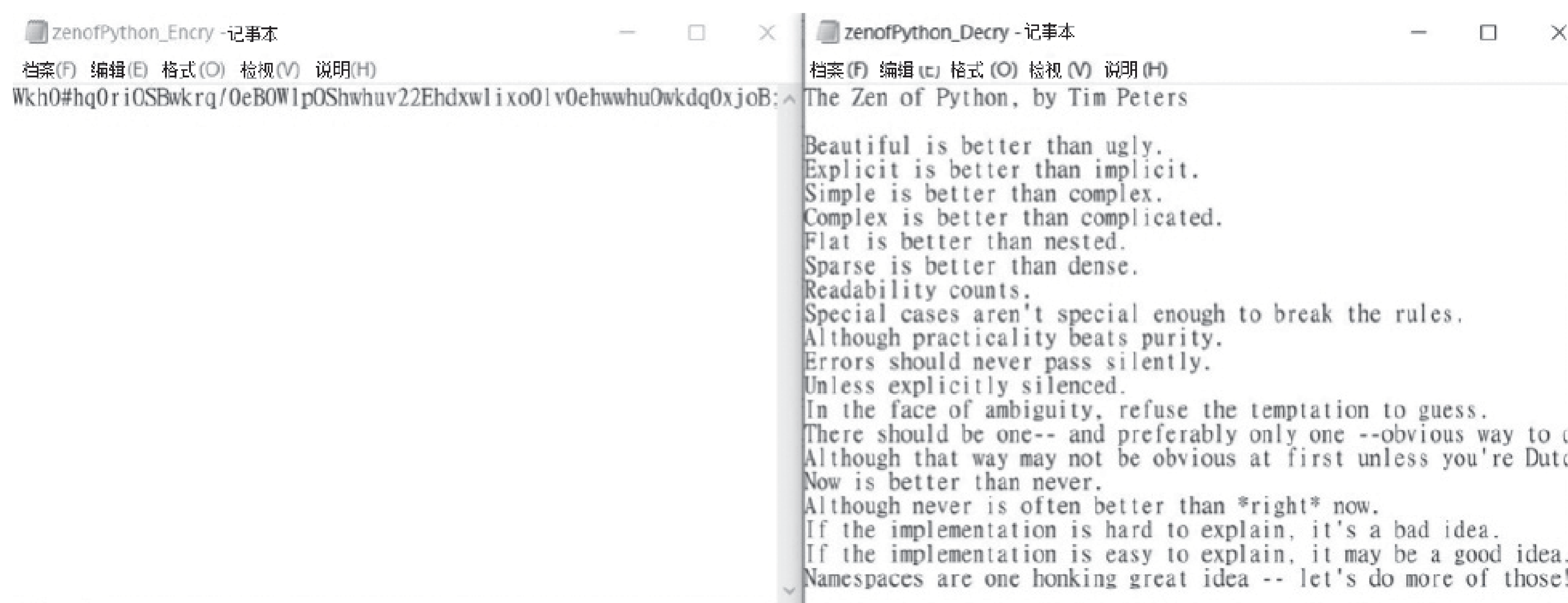
13. 请参考 ch14_43.py 执行解压缩文件，读者必须从屏幕输入要解压缩的文件，以及存放的目录位置。(14-6 节)

```
====================== RESTART: D:\Python\ex\ex14_13.py ======================
请输入要解压缩的目录 : zip14_12.zip
请输入存放解压缩的目录 : result
```

14. 请扩充设计 ch14_51.py，这个程序将所有出现的单词按从多到少打印出来。(14-9 节)

```
====================== RESTART: D:/Python/ex/ex14_14.py ======================
ding : 4
are : 4
bells : 2
sleeping : 2
morning : 2
dong : 2
ringing : 2
john : 2
you : 2
brother : 2
```

15. 为 ch14_52.py 所加密的字符串存入 zenofPython_Encry.txt，同时解密所加密的字符串，最后将解密的结果存入 zenofPython_Decry.txt，然后打开文件观察执行结果。



Python

数据科学零基础一本通

下册

洪锦魁◎著

- 人工智能基础知识
- Python彩蛋
- bytes数据、编码、译码
- 用经纬度计算城市间的距离
- 用数学方法计算圆周率
- 生成式
- lambda、map( )、reduce( )
- 文件加密与解密
- 文件读取、写入与目录管理
- 程序异常处理与程序日志
- 正则表达式
- 递归式观念与碎形
- 图像、QR code、文字辨识
- GUI、动画、游戏、小计算器
- CSV和JSON文件
- 词云设计
- 股市数据读取与图表制作
- 基础统计、线性代数
- matplotlib中英文图表绘制
- Numpy、Scipy、Pandas

800多个程序实例
400多个模块
200多道实践习题

清华大学出版社

Python

数据科学零基础一本通

下册

洪锦魁◎著

清华大学出版社
北　京

内 容 简 介

这是一本专为没有编程基础的读者编写的 Python 入门书籍，全书包含 800 多个程序实例及 200 多道实践习题，一步一步详细讲解 Python 语法的基础知识，同时也将应用范围拓展至图形界面设计、影像处理、图表绘制、文字识别、词云、股市资料摘取与图表制作、线性代数、基础统计以及与数据科学相关的 Numpy、Scipy、Pandas。Python 是一门非常灵活的编程语言，本书特色在于对 Python 的基础知识与应用辅以大量实例进行讲解，读者可以通过这些程序实例事半功倍地学会 Python。

图书在版编目（CIP）数据

Python数据科学零基础一本通 / 洪锦魁著. —北京：清华大学出版社，2020.2
ISBN 978-7-302-54539-2

Ⅰ. ①P…　Ⅱ. ①洪…　Ⅲ. ①软件工具—程序设计　Ⅳ. ①TP311.561

中国版本图书馆 CIP 数据核字（2019）第 290377 号

责任编辑： 张　敏　薛　阳
封面设计： 杨玉兰
责任校对： 徐俊伟
责任印制： 丛怀宇

出版发行： 清华大学出版社
网　　址： http://www.tup.com.cn，http://www.wqbook.com
地　　址： 北京清华大学学研大厦 A 座　　**邮　　编：** 100084
社 总 机： 010-62770175　　**邮　　购：** 010-62786544
投稿与读者服务： 010-62776969，c-service@tup.tsinghua.edu.cn
质 量 反 馈： 010-62772015，zhiliang@tup.tsinghua.edu.cn
印 装 者： 三河市铭诚印务有限公司
经　　销： 全国新华书店
开　　本： 170mm×240mm　　**印　　张：** 48.75　　**字　　数：** 1278 千字
版　　次： 2020 年 4 月第 1 版　　**印　　次：** 2020 年 4 月第 1 次印刷
定　　价： 129.00 元（上、下册）

产品编号：085277-01

序

多次与教育界的朋友相聚，谈到计算机语言的发展趋势时，大家一致认为 Python 是当今最重要的计算机语言。许多知名公司，例如 Google、Facebook 等皆已将 Python 列为必备计算机语言。许多人想学 Python，市面上的书也不少，但书中对 Python 语法的讲解并不完整，造成读者学习上的障碍，读者读完一本 Python 书籍，仍然看不懂专家写的 Python 程序。因此，笔者决定撰写一本用丰富、实用、有趣的实例完整且深入讲解 Python 语法的入门书籍。

Python 以简洁著名，语法非常灵活，同时拥有丰富、实用的模块。本书除了以实例解说 Python 语法，还会穿插讲解各种模块，以帮助读者更灵活地掌握 Python。此外，笔者也尝试在书中穿插基本的科学、数学、统计与人工智能的基础知识，帮助读者为进一步的学习打下扎实的基础。

本书包含 800 多个程序实例，搭配 400 多个模块，并辅以 200 多道实践习题，细致讲解 Python 语法。本书也会说明下列知识与应用：

- 人工智能基础知识；
- Python 彩蛋；
- 从 bytes 数据、编码（encode）、译码（decode）说起，到精通列表（list）、元组（tuple）、字典（dict）、集合（set）；
- 从小型列表、元组、字典到大型数据资料的建立；
- 生成式（generator）建立 Python 数据结构；
- 在坐标轴内计算任意两点之间的距离，同时解说与人工智能的关联；
- 用经纬度计算地球任意两座城市之间的距离，学习取得地球任意位置的经纬度；
- 用莱布尼茨公式、尼拉卡莎级数、蒙特卡罗模拟计算圆周率；
- 讲解基础函数观念，也深入到嵌套、closure、lambda、Decorator 等高阶应用；
- 对 map( ) 和 reduce( ) 进行完整解说，并进一步配合 lambda 解说高级应用；
- 建立类别的同时深入讲解装饰器 @property、@classmethod、@staticmethod 与类别特殊属性与方法；

- 设计与应用自己设计的模块、活用外部模块（module）；
- 赌场骗局；
- 自己设计加密与解密程序；
- Python 的输入与输出；
- 文件压缩与解压缩；
- 程序除错与异常处理；
- 文件读取与目录管理；
- 剪贴板应用；
- 正则表达式；
- 递归式观念与碎形 Fractal；
- 图像处理与文字辨识，更进一步说明计算机储存图像的方法；
- 基本与进阶 QR code 制作；
- 词云（Word Cloud）设计；
- GUI 设计：设计小计算器；
- 动画与游戏；
- matplotlib 中英文图表绘制；
- 说明 CSV 和 JSON 文件；
- 股市数据读取与图表制作；
- Python 解线性代数；
- Python 解联立方程式；
- Python 执行数据分析；
- 科学计算与数据分析 Numpy、Scipy、Pandas。

笔者编写过许多计算机领域的著作，本书将沿袭笔者以往著作的特色，程序实例丰富。相信读者通过学习本书内容，一定可以快速精通 Python。笔者虽力求完美，但是书中不足与疏漏在所难免，请不吝指正。

洪锦魁

2019.10.31

目　　录

第 21 章 JSON 资料

第 22 章 使用 Python 处理 CSV 文件

15

第 15 章

程序除错与异常处理

本章摘要

15-1 程序异常
15-2 设计多组异常处理程序
15-3 丢出异常
15-4 记录 Traceback 字符串
15-5 finally
15-6 程序断言 assert
15-7 程序日志模块 logging
15-8 程序除错的典故

15-1 程序异常

有时也可以将**程序错误**（error）称作**程序异常**（exception），每一个写程序的人都会碰上程序错误，过去碰上这类情况程序将终止执行，同时出现错误消息，错误消息的内容通常是显示Traceback，然后列出异常报告。Python 提供了可以**捕捉异常和撰写异常处理程序**的功能，当发生异常被我们捕捉时会去执行异常处理程序，然后程序就可以继续执行。

15-1-1 一个除数为 0 的错误

本节将以一个除数为 0 的错误开始说明。

程序实例 ch15_1.py：建立一个除法运算的函数，这个函数将接收两个参数，然后用第一个参数除以第二个参数。

```
1  # ch15_1.py
2  def division(x, y):
3      return x / y
4
5  print(division(10, 2))      # 列出10/2
6  print(division(5, 0))       # 列出5/0
7  print(division(6, 3))       # 列出6/3
```

执行结果

```
==================== RESTART: D:\Python\ch15\ch15_1.py ====================
5.0
Traceback (most recent call last):
  File "D:\Python\ch15\ch15_1.py", line 6, in <module>
    print(division(5, 0))       # 列出5/0
  File "D:\Python\ch15\ch15_1.py", line 3, in division
    return x / y
ZeroDivisionError: division by zero
```

上述程序在执行第 5 行时，一切都正常。但是到了执行第 6 行时，因为第 2 个参数是 0，导致发生 ZeroDivisionError: division by zero 的错误，所以整个程序就执行终止了。其实对于上述程序而言，若是程序可以执行第 7 行，是可以正常得到执行结果的，可是程序第 6 行已经造成程序终止了，所以无法执行第 7 行。

15-1-2 撰写异常处理程序 try – except

这一节将讲解如何捕捉异常与设计异常处理程序，发生异常被捕捉时程序会执行异常处理程序，然后跳开异常位置，再继续往下执行。这时要使用 try – except 指令，语法格式如下。

```
try:
    指令              # 预先设想可能引发错误异常的指令
except 异常对象：      # 若以 ch15_1.py 而言，异常对象就是指 ZeroDivisionError
    异常处理程序       # 通常是指出异常原因，方便修改
```

上述语句会执行 try: 下面的**指令**，如果正常则跳离 except 部分，如果**指令**有异常，则检查此异常是否是**异常对象**所指的错误，如果是代表异常被捕捉了，则执行此**异常对象**下面的异常处理程序。

程序实例 ch15_2.py：重新设计 ch15_1.py，增加异常处理程序。

```
1  # ch15_2.py
2  def division(x, y):
3      try:                          # try - except指令
4          return x / y
5      except ZeroDivisionError:     # 除数为0时执行
6          print("除数不可为0")
7
8  print(division(10, 2))            # 列出10/2
9  print(division(5, 0))             # 列出5/0
10 print(division(6, 3))             # 列出6/3
```

执行结果

```
==================== RESTART: D:\Python\ch15\ch15_2.py ====================
5.0
除数不可为0
None
2.0
>>>
```

上述程序执行第 8 行时，会将参数（10, 2）带入 division( ) 函数，由于执行 try 的指令的“x / y”没有问题，所以可以执行“return x / y”，这时 Python 将跳过 except 指令。当程序执行第 9 行时，会将参数（5, 0）带入 division( ) 函数，由于执行 try 的指令的“x / y”产生了除数为 0 的 ZeroDivisionError 异常，这时 Python 会查找是否有处理这类异常的 except ZeroDivisionError 存在，如果有就表示此异常被捕捉，就去执行相关的错误处理程序，此例是执行第 6 行，打印出“除数不可为 0”的错误。函数回返然后打印出结果 None。None 是一个对象，表示结果不存在，最后返回程序第 10 行，继续执行相关指令。

从上述可以看到，程序增加了 try – except 后，若是异常被 except 捕捉，出现的异常消息就比较友善了，同时不会有程序中断的情况发生。

特别需留意的是，在 try – except 的使用中，如果在 try: 后面的**指令**产生异常时，这个异常不是我们设计的 except **异常对象**，表示异常没被捕捉到，这时程序依旧会像 ch15_1.py 一样，直接出现错误消息，然后程序终止。

程序实例 ch15_2_1.py：重新设计 ch15_2.py，但是程序第 9 行使用字符调用除法运算，造成程序异常。

```
1  # ch15_2_1.py
2  def division(x, y):
3      try:                          # try - except指令
4          return x / y
5      except ZeroDivisionError:     # 除数为0时执行
6          print("除数不可为0")
7
8  print(division(10, 2))            # 列出10/2
9  print(division('a', 'b'))         # 列出'a' / 'b'
10 print(division(6, 3))             # 列出6/3
```

执行结果

```
=================== RESTART: D:/Python/ch15/ch15_2_1.py ===================
5.0
Traceback (most recent call last):
  File "D:/Python/ch15/ch15_2_1.py", line 9, in <module>
    print(division('a', 'b'))        # 列出'a' / 'b'
  File "D:/Python/ch15/ch15_2_1.py", line 4, in division
    return x / y
TypeError: unsupported operand type(s) for /: 'str' and 'str'
>>>
```

由上述执行结果可以看到异常原因是 TypeError，由于我们在程序中没有设计 except TypeError 的异常处理程序，所以程序会终止执行。更多相关处理将在 15-2 节说明。

15-1-3 try – except – else

Python 在 try – except 中又增加了 else 指令，这个指令存放的主要目的是 try 内的**指令**正确时，可以执行 else 内的指令区块，我们可以将这部分指令区块称为**正确处理程序**，这样可以增加程序的可读性。此时语法格式如下。

```
try:
    指令                # 预先设想可能引发异常的指令
except 异常对象：       # 若以 ch15_1.py 而言，异常对象就是指 ZeroDivisionError
    异常处理程序        # 通常是指出异常原因，方便修改
else:
    正确处理程序        # 如果指令正确时执行此区块指令
```

程序实例 ch15_3.py：使用 try – except – else 重新设计 ch15_2.py。

```
1  # ch15_3.py
2  def division(x, y):
3      try:                          # try - except指令
4          ans =  x / y
5      except ZeroDivisionError:     # 除数为0时执行
6          print("除数不可为0")
7      else:
8          return ans                # 返回正确的执行结果
9
10 print(division(10, 2))            # 列出10/2
11 print(division(5, 0))             # 列出5/0
12 print(division(6, 3))             # 列出6/3
```

执行结果 与 ch15_2.py 相同。

15-1-4 找不到文件的错误 FileNotFoundError

程序设计时另一个常常发生的异常是打开文件时找不到该文件，这时会产生 FileNotFoundError 异常。

程序实例 ch15_4.py：打开一个不存在的文件 ch15_4.txt 产生异常的实例，这个程序会有一个异常处理程序，列出文件不存在。如果文件存在则打印文件内容。

```
1  # ch15_4.py
2
3  fn = 'ch15_4.txt'                 # 设置要打开的文件
4  try:
5      with open(fn) as file_Obj:    # 用默认mode=r打开文件,返回调用对象file_Obj
6          data = file_Obj.read()    # 读取文件到变量data
7  except FileNotFoundError:
8      print("找不到 %s 文件" % fn)
9  else:
10     print(data)                   # 输出变量data相当于输出文件
```

执行结果

```
==================== RESTART: D:\Python\ch15\ch15_4.py ====================
找不到 ch15_4.txt文件
>>>
```

本文件夹 ch15 内有 ch15_5.txt，相同的程序只是第 3 行打开的文件不同，将可以获得打印出 ch15_5.txt。

程序实例 ch15_5.txt：与 ch15_4.txt 内容基本上相同，只是打开的文件不同。

```
3  fn = 'ch15_5.txt'                    # 设置要打开的文件
```

执行结果

```
==================== RESTART: D:\Python\ch15\ch15_5.py ====================
DeepMind Co.
I like DeepMind
Deep Learning

>>>
```

15-1-5　分析单一文件的字数

有时候在读一篇文章时，可能会想知道这篇文章的字数，这时可以采用下列方式分析。在正式分析前，可以先来看一个简单的程序应用。如果忘记 split( ) 方法，可复习 6-9-6 节。

程序实例 ch15_6.py：分析一个文件内有多少个单字。

```
1  # ch15_6.py
2
3  fn = 'ch15_6.txt'                          # 设置要打开的文件
4  try:
5      with open(fn) as file_Obj:             # 用默认mode=r打开文件,返回调用对象file_Obj
6          data = file_Obj.read()             # 读取文件到变量data
7  except FileNotFoundError:
8      print("找不到 %s 文件" % fn)
9  else:
10     wordList = data.split()                # 将文章转成列表
11     print(fn, " 文章的字数是 ", len(wordList))   # 打印文章字数
```

执行结果

```
==================== RESTART: D:\Python\ch15\ch15_6.py ====================
ch15_6.txt  文章的字数是  43
>>>
```

如果程序设计时需要计算某篇文章的字数，可以考虑将上述计算文章的字数处理成一个函数，这个函数的参数是文章的文件名，然后函数直接打印出文章的字数。

程序实例 ch15_7.py：设计一个计算文章字数的函数 wordsNum，只要传递文章文件名，就可以获得此篇文章的字数。

```
1   # ch15_7.py
2   def wordsNum(fn):
3       """适用英文文件，输入文章的文件名,可以计算此文章的字数"""
4       try:
5           with open(fn) as file_Obj:                # 用默认"r"返回调用对象file_Obj
6               data = file_Obj.read()                # 读取文件到变量data
7       except FileNotFoundError:
8           print("找不到 %s 文件" % fn)
9       else:
10          wordList = data.split()                   # 将文章转成列表
11          print(fn, " 文章的字数是 ", len(wordList))  # 打印文章字数
12
13  file = 'ch15_6.txt'                               # 设置要打开的文件
14  wordsNum(file)
```

执行结果 与 ch15_6.py 相同。

15-1-6 分析多个文件的字数

程序设计时可能需设计读取许多文件做分析，部分文件可能存在，部分文件可能不存在，这时就可以使用本节的概念做设计了。在接下来的程序实例分析中，笔者将要读取的文件名放在列表内，然后使用循环将文件分次传给程序实例 ch15_7.py 建立的 wordsNum 函数，如果文件存在将打印出字数，如果文件不存在将列出找不到此文件。

程序实例 ch15_8.py：分析 data1.txt、data2.txt、data3.txt 这 3 个文件的字数，同时笔者在 ch15 文件夹下没有放置 data2.txt，所以程序遇到分析此文件时，将列出找不到此文件。

```
1   # ch15_8.py
2   def wordsNum(fn):
3       """适用英文文件，输入文章的文件名,可以计算此文章的字数"""
4       try:
5           with open(fn) as file_Obj:                # 用默认"r"返回调用对象file_Obj
6               data = file_Obj.read()                # 读取文件到变量data
7       except FileNotFoundError:
8           print("找不到 %s 文件" % fn)
9       else:
10          wordList = data.split()                   # 将文章转成列表
11          print(fn, " 文章的字数是 ", len(wordList))  # 打印文章字数
12
13  files = ['data1.txt', 'data2.txt', 'data3.txt']   # 文件列表
14  for file in files:
15      wordsNum(file)
```

执行结果

```
==================== RESTART: D:\Python\ch15\ch15_8.py ====================
data1.txt  文章的字数是  43
找不到 data2.txt 文件
data3.txt  文章的字数是  39
>>>
```

15-2 设计多组异常处理程序

在程序实例 ch15_1.py、ch15_2.py 和 ch15_2_1.py 中，我们很清楚地了解了程序设计中有太多各种不可预期的异常发生，所以需要了解设计程序时可能需要同时设计多个异常处理程序。

15-2-1　常见的异常对象

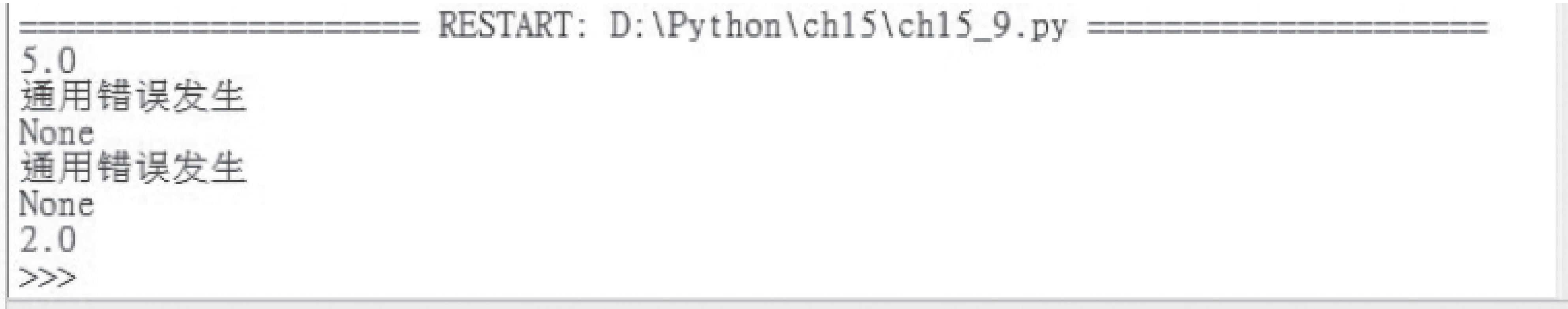
异常对象名称	说明
AttributeError	通常是指对象没有这个属性
Exception	一般错误都可使用
FileNotFoundError	找不到 open( ) 打开的文件
IOError	在输入或输出时发生错误
IndexError	索引超出范围区间
KeyError	在映射中没有这个键
MemoryError	需求内存空间超出范围
NameError	对象名称未声明
SyntaxError	语法错误
SystemError	直译器的系统错误
TypeError	数据类型错误
ValueError	传入无效参数
ZeroDivisionError	除数为 0

在 ch15_2_1.py 的程序应用中可以发现，异常发生时如果 except 设置的异常对象不是发生的异常，相当于 except 没有捕捉到异常，所设计的异常处理程序变成无效的异常处理程序。Python 提供了一个通用型的异常对象 Exception，它可以捕捉各式的基础异常。

程序实例 ch15_9.py：重新设计 ch15_2_1.py，异常对象设为 Exception。

```
1   # ch15_9.py
2   def division(x, y):
3       try:                          # try - except指令
4           return x / y
5       except Exception:             # 通用错误使用
6           print("通用错误发生")
7
8   print(division(10, 2))            # 列出10/2
9   print(division(5, 0))             # 列出5/0
10  print(division('a', 'b'))         # 列出'a' / 'b'
11  print(division(6, 3))             # 列出6/3
```

执行结果



```
==================== RESTART: D:\Python\ch15\ch15_9.py ====================
5.0
通用错误发生
None
通用错误发生
None
2.0
>>>
```

从上述可以看到，第 9 行**除数为 0** 或是第 10 行**字符相除**所产生的异常都可以使用 except Exception 予以捕捉，然后执行异常处理程序。甚至这个通用型的异常对象也可以应用于取代 FileNotFoundError 异常对象。

程序实例 ch15_10.py：使用 Exception 取代 FileNotFoundError，重新设计 ch15_8.py。

```
7       except Exception:
8           print("Exception找不到 %s 文件" % fn)
```

执行结果

```
===================== RESTART: D:\Python\ch15\ch15_10.py =====================
data1.txt  文章的字数是  43
Exception找不到 data2.txt 文件
data3.txt  文章的字数是  39
>>>
```

15-2-2 设计捕捉多个异常

在 try- except 的使用中，可以设计多个 except 捕捉多种异常，语法如下。

```
try:
    指令                      # 预先设想可能引发错误异常的指令
except 异常对象 1:            # 如果指令发生异常对象 1 执行
    异常处理程序 1
except 异常对象 2:            # 如果指令发生异常对象 2 执行
    异常处理程序 2
```

当然也可以视情况设计更多异常处理程序。

程序实例 ch15_11.py：重新设计 ch15_9.py 捕捉两个异常对象，可参考第 5 和 7 行。

```
1  # ch15_11.py
2  def division(x, y):
3      try:                          # try - except指令
4          return x / y
5      except ZeroDivisionError:     # 除数为0使用
6          print("除数为0发生")
7      except TypeError:             # 数据类型错误
8          print("使用字符做除法运算异常")
9
10 print(division(10, 2))            # 列出10/2
11 print(division(5, 0))             # 列出5/0
12 print(division('a', 'b'))         # 列出'a' / 'b'
13 print(division(6, 3))             # 列出6/3
```

执行结果 与 ch15_9.py 相同。

15-2-3 使用一个 except 捕捉多个异常

Python 也允许设计一个 except 捕捉多个异常，语法如下。

```
try:
    指令                                        # 预先设想可能引发错误异常的指令
except （异常对象 1，异常对象 2，…）：          # 指令发生其中所列异常对象执行
    异常处理程序
```

程序实例 ch15_12.py：重新设计 ch15_11.py，用一个 except 捕捉两个异常对象，下列程序读者需留意第 5 行的 except 的写法。

```
h15_12.py
 division(x, y):
 try:                                           # try - except指令
     return x / y
 except (ZeroDivisionError, TypeError):         # 两个异常
     print("除数为0发生 或 使用字符做除法运算异常")

nt(division(10, 2))                             # 列出10/2
nt(division(5, 0))                              # 列出5/0
nt(division('a', 'b'))                          # 列出'a' / 'b'
nt(division(6, 3))                              # 列出6/3
```

执行结果

```
==================== RESTART: D:\Python\ch15\ch15_12.py ====================
5.0
除数为0发生 或 使用字符做除法运算异常
None
除数为0发生 或 使用字符做除法运算异常
None
2.0
>>>
```

15-2-4　处理异常但是使用 Python 内建的错误消息

在先前所有实例中，当发生异常同时被捕捉时都是使用我们自建的异常处理程序，Python 也支持发生异常时使用系统内建的异常处理消息，语法格式如下。

```
try:
    指令                                # 预先设想可能引发错误异常的指令
except 异常对象 as e:                   # 使用 as e
    print(e)                            # 输出 e
```

上述 e 是系统内建的异常处理消息，可以是任意字符，笔者此处使用 e 是因为可代表 error 的内涵。当然上述 except 语法也接收同时处理多个异常对象，可参考下列程序实例第 5 行。

程序实例 ch15_13.py：重新设计 ch15_12.py，使用 Python 内建的错误消息。

```
1  # ch15_13.py
2  def division(x, y):
3      try:                                              # try - except指令
4          return x / y
5      except (ZeroDivisionError, TypeError) as e:       #两个异常
6          print(e)
7
8  print(division(10, 2))                                # 列出10/2
9  print(division(5, 0))                                 # 列出5/0
10 print(division('a', 'b'))                             # 列出'a' / 'b'
11 print(division(6, 3))                                 # 列出6/3
```

执行结果

```
==================== RESTART: D:\Python\ch15\ch15_13.py ====================
5.0
division by zero
None
unsupported operand type(s) for /: 'str' and 'str'
None
2.0
```

上述执行结果的错误消息都是 Python 内部的错误消息。

15-2-5　捕捉所有异常

程序设计中许多异常是我们不可预期的，很难一次设想周到，Python 也提供了语法让我们可以一次捕捉所有异常，如下。

```
try:
    指令                                # 预先设想可能引发错误异常的指令
except:                                 # 捕捉所有异常
    异常处理程序                        # 通常是 print 输出异常说明
```

程序实例 ch15_14.py：一次捕捉所有异常的设计。

```
1  # ch15_14.py
2  def division(x, y):
3      try:                         # try - except指令
4          return x / y
5      except:                      # 捕捉所有异常
6          print("异常发生")
7
8  print(division(10, 2))           # 列出10/2
9  print(division(5, 0))            # 列出5/0
10 print(division('a', 'b'))        # 列出'a' / 'b'
11 print(division(6, 3))            # 列出6/3
```

执行结果

```
==================== RESTART: D:\Python\ch15\ch15_14.py ====================
5.0
异常发生
None
异常发生
None
2.0
>>>
```

15-3 丢出异常

前面所介绍的异常都是 Python 直译器发现异常时，自行丢出异常对象，如果我们不处理程序就终止执行，使用 try – except 处理程序可以在异常中恢复执行。这一节要探讨的是，设计程序时如果发生某些状况，我们自己将它定义为异常然后丢出异常消息，程序停止正常往下执行，同时让程序跳到自己设计的 except 去执行。它的语法如下：

```
raise Exception('msg')              # 调用 Exception,msg 是传递错误消息
…
…
try:
    指令
except Exception as err:            # err 是任意取的变量名称，内容是 msg
    print("message", + str(err))    # 打印错误消息
```

程序实例 ch15_15.py：目前有些金融机构在客户建立网络账号时，会要求密码长度必须为 5 ～ 8 个字符，接下来设计一个程序，这个程序内有 passWord( ) 函数，这个函数会检查密码长度，如果长度小于 5 或是长度大于 8 都抛出异常。在第 11 行会有一系列密码供测试，然后以循环方式执行检查。

```
1  # ch15_15.py
2  def passWord(pwd):
3      """检查密码长度必须是5到8个字符"""
4      pwdlen = len(pwd)                        # 密码长度
5      if pwdlen < 5:                           # 密码长度不足
6          raise Exception('密码长度不足')
7      if pwdlen > 8:                           # 密码长度太长
8          raise Exception('密码长度太长')
9      print('密码长度正确')
10
11 for pwd in ('aaabbbccc', 'aaa', 'aaabbb'):   # 测试系列密码值
12     try:
13         passWord(pwd)
14     except Exception as err:
15         print("密码长度检查异常发生: ", str(err))
```

执行结果

```
==================== RESTART: D:\Python\ch15\ch15_15.py ====================
密码长度检查异常发生:  密码长度太长
密码长度检查异常发生:  密码长度不足
密码长度正确
>>>
```

上述语句当密码长度不足或密码长度太长时，都会抛出异常，这时 passWord( ) 函数返回的是 Exception 对象（第 6 和 8 行），原先 Exception( ) 内的字符串（'密码长度不足' 或 '密码长度太长'）会通过第 14 行传给 err 变量，然后执行第 15 行内容。

15-4 记录 Traceback 字符串

相信读者学习至今，已经经历了许多程序设计的错误，每次出错误屏幕上都会出现 Traceback 字符串，在这个字符串中指出程序错误的原因。例如，请参考程序实例 ch15_2_1.py 的执行结果，该程序使用 Traceback 列出了错误。

如果导入 traceback 模块，就可以使用 traceback.format_exc( ) 记录这个 Traceback 字符串。

程序实例 ch15_16.py：重新设计程序实例 ch15_15.py，增加记录 Traceback 字符串，这个记录将被记录在 errch15_16.txt 内。

```
1   # ch15_16.py
2   import traceback                              # 导入traceback
3
4   def passWord(pwd):
5       """检查密码长度必须是5到8个字符"""
6       pwdlen = len(pwd)                         # 密码长度
7       if pwdlen < 5:                            # 密码长度不足
8           raise Exception('The length of pwd is too short')
9       if pwdlen > 8:                            # 密码长度太长
10          raise Exception('The length of pwd is too long')
11      print('密码长度正确')
12
13  for pwd in ('aaabbbccc', 'aaa', 'aaabbb'):   # 测试系列密码值
14      try:
15          passWord(pwd)
16      except Exception as err:
17          errlog = open('errch15_16.txt', 'a')    # 打开错误文件
18          errlog.write(traceback.format_exc())    # 写入错误文件
19          errlog.close()                          # 关闭错误文件
20          print("将Traceback写入错误文件errch15_16.txt完成")
21          print("密码长度检查异常发生: ", str(err))
```

执行结果

```
==================== RESTART: D:/Python/ch15/ch15_16.py ====================
将Traceback写入错误文件errch15_16.txt完成
密码长度检查异常发生:  The length of pwd is too long
将Traceback写入错误文件errch15_16.txt完成
密码长度检查异常发生:  The length of pwd is too short
密码长度正确
>>>
```

如果使用记事本打开 errch15_16.txt，可以得到下列结果。

```
Traceback (most recent call last):
  File "D:/Python/ch15/ch15_16.py", line 15, in <module>
    passWord(pwd)
  File "D:/Python/ch15/ch15_16.py", line 10, in passWord
    raise Exception('密码长度太长')
Exception: 密码长度太长
Traceback (most recent call last):
  File "D:/Python/ch15/ch15_16.py", line 15, in <module>
    passWord(pwd)
  File "D:/Python/ch15/ch15_16.py", line 8, in passWord
    raise Exception('密码长度不足')
Exception: 密码长度不足
```

上述程序第 17 行使用‘a’附加文件方式打开文件，主要是程序执行期间可能有多个错误，为了记录所有错误所以使用这种方式打开文件。上述程序最关键的地方是第 17 ～ 19 行，在这里打开了记录错误的 errch15_16.txt 文件，然后将错误写入此文件，最后关闭此文件。这个程序纪录的错误是我们抛出的异常错误，其实在 15-1 节和 15-2 节中就设计了异常处理程序，避免错误造成程序中断，实际上 Python 还是有记录错误，可参考下一个实例。

程序实例 ch15_17.py：重新设计 ch15_14.py，主要是将程序异常的消息保存在 errch15_17.txt 文件内，本程序的重点是第 8 ～ 10 行。

```
1   # ch15_17.py
2   import traceback
3
4   def division(x, y):
5       try:                                  # try - except指令
6           return x / y
7       except:                               # 捕捉所有异常
8           errlog = open('errch15_17.txt', 'a')  # 打开错误文件
9           errlog.write(traceback.format_exc())  # 写入错误文件
10          errlog.close()                    # 关闭错误文件
11          print("将Traceback写入错误文件errch15_17.txt完成")
12          print("异常发生")
13
14  print(division(10, 2))                    # 列出10/2
15  print(division(5, 0))                     # 列出5/0
16  print(division('a', 'b'))                 # 列出'a' / 'b'
17  print(division(6, 3))                     # 列出6/3
```

执行结果

```
==================== RESTART: D:\Python\ch15\ch15_17.py ====================
5.0
将Traceback写入错误文件errch15_17.txt完成
异常发生
None
将Traceback写入错误文件errch15_17.txt完成
异常发生
None
2.0
>>>
```

如果使用记事本打开 errch15_17.txt，可以得到下列结果。

```
errch15_17 - 记事本
档案(F) 编辑(E) 格式(O) 检视(V) 说明(H)
Traceback (most recent call last):
  File "D:/Python/ch15/ch15_17.py", line 6, in division
    return x / y
ZeroDivisionError: division by zero
Traceback (most recent call last):
  File "D:/Python/ch15/ch15_17.py", line 6, in division
    return x / y
TypeError: unsupported operand type(s) for /: 'str' and 'str'
```

15-5 finally

Python 的关键词 finally 是和 try 配合使用的，在 try 之后可以有 except 或 else，这个 finally 关键词必须放在 except 和 else 之后，同时不论是否有异常发生一定会执行这个 finally 内的程序代码。其功能主要是用在 Python 程序与数据库连接时，输出连接相关信息。

程序实例 ch15_18.py：重新设计 ch15_14.py，增加 finally 关键词。

```
1   # ch15_18.py
2   def division(x, y):
3       try:                            # try - except指令
4           return x / y
5       except:                         # 捕捉所有异常
6           print("异常发生")
7       finally:                        # 离开函数前先执行此程序代码
8           print("阶段任务完成")
9
10  print(division(10, 2),"\n")         # 列出10/2
11  print(division(5, 0),"\n")          # 列出5/0
12  print(division('a', 'b'),"\n")      # 列出'a' / 'b'
13  print(division(6, 3),"\n")          # 列出6/3
```

执行结果

```
===================== RESTART: D:\Python\ch15\ch15_18.py =====================
阶段任务完成
5.0

异常发生
阶段任务完成
None

异常发生
阶段任务完成
None

阶段任务完成
2.0

>>>
```

上述程序执行时，如果没有发生异常，程序会先输出字符串“**阶段任务完成**”，然后返回主程序，输出 division( ) 的返回值。如果程序有异常会先输出字符串“**异常发生**”，再执行 finally 的程序代码输出字符串“**阶段任务完成**”，然后返回主程序，输出“None”。

15-6 程序断言 assert

15-6-1　设计断言

Python 的 assert 关键词主要功能是协助程序设计师在程序设计阶段，对整个程序的执行状态做一个全面性的安全检查，以确保程序不会发生语意上的错误。例如，在第 12 章设计银行的存款程序时，没有考虑到存款或提款是负值的问题，也没有考虑到如果提款金额大于存款金额的情况。

程序实例 ch15_19.py：重新设计 ch12_4.py，这个程序主要是将第 22 行的存款金额改为 −300 和第 24 行取款金额大于存款金额，接着观察执行结果。

```
1  # ch15_19.py
2  class Banks():
3      # 定义银行类
4      title = 'Taipei Bank'                       # 定义属性
5      def __init__(self, uname, money):           # 初始化方法
6          self.name = uname                       # 设置存款者名字
7          self.balance = money                    # 设置所存的钱
8
9      def save_money(self, money):                # 设计存款方法
10         self.balance += money                   # 执行存款
11         print("存款 ", money, " 完成")           # 打印存款完成
12
13     def withdraw_money(self, money):            # 设计取款方法
14         self.balance -= money                   # 执行取款
15         print("取款 ", money, " 完成")           # 打印取款完成
16
17     def get_balance(self):                      # 获得存款余额
18         print(self.name.title(), " 目前余额: ", self.balance)
19
20 hungbank = Banks('hung', 100)                   # 定义对象hungbank
21 hungbank.get_balance()                          # 获得存款余额
22 hungbank.save_money(-300)                       # 存款-300元
23 hungbank.get_balance()                          # 获得存款余额
24 hungbank.withdraw_money(700)                    # 取款700元
25 hungbank.get_balance()                          # 获得存款余额
```

执行结果

```
==================== RESTART: D:\Python\ch15\ch15_19.py ====================
Hung  目前余额:  100
存款  -300  完成
Hung  目前余额:  -200
取款  700  完成
Hung  目前余额:  -900
>>>
```

上述程序语法上是没有错误，但是犯了两个程序语意上的设计错误，分别是存款金额出现了负值和取款金额大于存款金额的问题。所以我们发现存款余额出现了负值 -200 和 -900 的情况。接下来将讲解如何解决上述问题。

断言（assert）主要功能是确保程序执行的某个阶段，必须符合一定的条件，如果不符合这个条件时程序将主动**抛出异常**，让**程序终止**，同时程序主动打印出异常原因，以方便程序设计师排错。它的语法格式如下：

assert **条件**，'字符串'

上述语法的意义是程序执行至此阶段时测试**条件**，如果条件响应是 True，程序不理会逗号“,”右边的**字符串**正常往下执行。如果条件响应是 False，程序终止同时将逗号“,”右边的**字符串**输出到 Traceback 的字符串内。对上述程序 ch15_19.py 而言，很明显重新设计 ch15_20.py 时必须让 assert 关键词做下列两件事。

（1）确保存款与取款金额是正值，否则输出错误，可参考第 10 和 15 行。

（2）确保取款金额小于等于存款金额，否则输出错误，可参考第 16 行。

程序实例 ch15_20.py：重新设计 ch15_19.py，在这个程序第 27 行先测试存款金额小于 0 的状况。

```
1   # ch15_20.py
2   class Banks():
3       # 定义银行类别
4       title = 'Taipei Bank'                    # 定义属性
5       def __init__(self, uname, money):        # 初始化方法
6           self.name = uname                    # 设置存款者名字
7           self.balance = money                 # 设置所存的钱
8
9       def save_money(self, money):             # 设计存款方法
10          assert money > 0, '存款money必须大于0'
11          self.balance += money                # 执行存款
12          print("存款 ", money, " 完成")        # 打印存款完成
13
14      def withdraw_money(self, money):         # 设计取款方法
15          assert money > 0, '取款money必须大于0'
16          assert money <= self.balance, '存款金额不足'
17          self.balance -= money                # 执行取款
18          print("取款 ", money, " 完成")        # 打印取款完成
19
20      def get_balance(self):                   # 获得存款余额
21          print(self.name.title(), " 目前余额: ", self.balance)
22
23  hungbank = Banks('hung', 100)                # 定义对象hungbank
24  hungbank.get_balance()                       # 获得存款余额
25  hungbank.save_money(300)                     # 存款300元
26  hungbank.get_balance()                       # 获得存款余额
27  hungbank.save_money(-300)                    # 存款-300元
28  hungbank.get_balance()                       # 获得存款余额
```

执行结果

```
==================== RESTART: D:\Python\ch15\ch15_20.py ====================
Hung  目前余额:  100
存款  300  完成
Hung  目前余额:  400
Traceback (most recent call last):
  File "D:\Python\ch15\ch15_20.py", line 27, in <module>
    hungbank.save_money(-300)                    # 存款-300元
  File "D:\Python\ch15\ch15_20.py", line 10, in save_money
    assert money > 0, '存款money必须大于0'
AssertionError: 存款money必须大于0
>>>
```

上述执行结果很清楚，程序第 27 行将存款金额设为负值 -300 时，调用 save_money( ) 方法，结果在第 10 行的 assert 断言处出现 False，所以设置的错误消息‘**存款必须大于 0**’的字符串被打印出来，这种设计方便我们在真实的环境做最后的程序语意检查。

程序实例 ch15_21.py：重新设计 ch15_20.py，这个程序测试了当取款金额大于存款金额的状况，可参考第 27 行，下面只列出主程序内容。

```
23  hungbank = Banks('hung', 100)                # 定义对象hungbank
24  hungbank.get_balance()                       # 获得存款余额
25  hungbank.save_money(300)                     # 存款300元
26  hungbank.get_balance()                       # 获得存款余额
27  hungbank.withdraw_money(700)                 # 取款700元
28  hungbank.get_balance()                       # 获得存款余额
```

执行结果

```
==================== RESTART: D:\Python\ch15\ch15_21.py ====================
Hung  目前余额:  100
存款  300  完成
Hung  目前余额:  400
Traceback (most recent call last):
  File "D:\Python\ch15\ch15_21.py", line 27, in <module>
    hungbank.withdraw_money(700)                 # 取款700元
  File "D:\Python\ch15\ch15_21.py", line 16, in withdraw_money
    assert money <= self.balance, '存款金额不足'
AssertionError: 存款金额不足
>>>
```

上述当取款金额大于存款金额时，这个程序将造成第 16 行的 assert 断言条件是 False，所以触发了打印‘**存款金额不足**’的消息。由上述的执行结果，我们就可以依据需要修正程序的内容。

15-6-2　停用断言

断言 assert 一般是用在程序开发阶段，如果整个程序设计好了以后，想要停用断言 assert，可以在 Windows 的命令提示环境（可参考附录 B-2-1），执行程序时使用“-O”选项停用断言。笔者在 Windows 8 操作系统上安装的 Python 3.62 版，在这个版本的 Python 安装路径 ~\Python\Python36-32 内有 python.exe 可以执行所设计的 Python 程序，以 ch15_21.py 为实例，如果要停用断言可以使用下列指令。

```
~\python.exe -O D:\Python\ch15\ch15_21.py
```

上述“~”代表安装 Python 的路径，若是以 ch15_21.py 为例，采用停用断言选项“-O”后，执行结果将看到不再有 Traceback 错误消息产生，因为断言被停用了。

```
命令提示字元
Microsoft Windows [版本 6.3.9600]
(c) 2013 Microsoft Corporation. 著作权所有，并保留一切权利。

C:\Users\Jiin-Kwei>C:\Users\Jiin-Kwei\AppData\Local\Programs\Python\Python36-32\
python.exe -O D:\Python\ch15\ch15_21.py
Hung  目前余额:  100
存款  300  完成
Hung  目前余额:  400
取款  700  完成
Hung  目前余额:  -300

C:\Users\Jiin-Kwei>

微軟注音 半 :
```

15-7　程序日志模块 logging

程序设计阶段难免会有错误产生，没有得到预期的结果，在产生错误期间到底发生了什么事情？程序代码执行顺序是否有误或是变量值如何变化？这些都是程序设计师想知道的事情。笔者过去碰上这方面的问题，常常是在程序代码几个重要节点增加 print( ) 函数输出关键变量，以了解程序的变化，程序修订完成后再将这几个 print( ) 删除，坦白地说是有一点儿麻烦。

Python 有**程序日志 logging** 功能，这个功能可以协助我们执行程序的除错，有了这个功能我们可以自行设置关键变量在每一个程序阶段的变化，由这个关键变量的变化可方便我们执行程序的除错，同时未来不想要显示这些关键变量数据时，可以不用删除，只要适度加上指令就可隐藏它们，这将是本节的主题。

15-7-1　logging 模块

Python 内有提供 logging 模块，这个模块有提供方法可以让我们使用**程序日志 logging** 功能，在使用前须先使用 import 导入此模块。

```
import logging
```

15-7-2　logging 的等级

logging 模块共分为 5 个等级，从最低到最高等级顺序如下。

1. DEBUG 等级

使用 logging.debug( ) 显示程序日志内容，所显示的内容是程序的小细节，最低层级的内容，感觉程序有问题时可使用它追踪关键变量的变化过程。

2. INFO 等级

使用 logging.info( ) 显示程序日志内容，所显示的内容是记录程序一般发生的事件。

3. WARNING 等级

使用 logging.warning( ) 显示程序日志内容，所显示的内容虽然不会影响程序的执行，但是未来可能导致问题的发生。

4. ERROR 等级

使用 logging.error( ) 显示程序日志内容，通常显示程序在某些状态将引发错误的缘由。

5. CRITICAL 等级

使用 logging.critical( ) 显示程序日志内容，这是最重要的等级，通常是显示将让整个系统 Down 掉或中断的错误。

程序设计时，可以使用下列函数设置显示信息的等级。

```
logging.basicConfig(level=logging.DEBUG)     # 假设是设置 DEBUG 等级
```

当设置 logging 为某一等级时，未来只有此等级或更高等级的 logging 会被显示。

程序实例 ch15_22.py：显示所有等级的 logging 消息。

```
1  # ch15_22.py
2  import logging
3
4  logging.basicConfig(level=logging.DEBUG)    # 等级是DEBUG
5  logging.debug('logging message, DEBUG')
6  logging.info('logging message, INFO')
7  logging.warning('logging message, WARNING')
8  logging.error('logging message, ERROR')
9  logging.critical('logging message, CRITICAL')
```

执行结果

```
===================== RESTART: D:/Python/ch15/ch15_22.py =====================
DEBUG:root:logging message, DEBUG
INFO:root:logging message, INFO
WARNING:root:logging message, WARNING
ERROR:root:logging message, ERROR
CRITICAL:root:logging message, CRITICAL
>>>
```

上述每一个输出前方有 DEBUG:root:（其他以此类推）前导消息，这是该 logging 输出模式默认的输出消息注明输出 logging 模式。

程序实例 ch15_23.py：显示 WARNING 等级或更高等级的输出。

```
1  # ch15_23.py
2  import logging
3
4  logging.basicConfig(level=logging.WARNING)    # 等级是WARNING
5  logging.debug('logging message, DEBUG')
6  logging.info('logging message, INFO')
7  logging.warning('logging message, WARNING')
8  logging.error('logging message, ERROR')
9  logging.critical('logging message, CRITICAL')
```

执行结果

```
==================== RESTART: D:/Python/ch15/ch15_23.py ====================
WARNING:root:logging message, WARNING
ERROR:root:logging message, ERROR
CRITICAL:root:logging message, CRITICAL
>>>
```

当我们设置 logging 的输出等级是 WARNING 时，较低等级的 logging 输出就被隐藏了。当了解了上述 logging 输出等级的特性后，笔者通常在设计大型程序时，程序设计初期阶段会将 logging 等级设为 DEBUG，如果确定程序大致没问题后，就将 logging 等级设为 WARNING，最后再设为 CRITICAL。这样就可以不用再像过去一样在程序设计初期使用 print( ) 记录关键变量的变化，当程序确定完成后，需要一个一个检查 print( ) 然后将它删除。

15-7-3　格式化 logging 消息输出 format

从 ch15_22.py 和 ch15_23.py 可以看到输出消息前方有前导输出消息，我们可以使用在 logging.basicConfig( ) 方法内增加 format 格式化输出消息为空字符串 '' 方式，取消显示前导输出消息。

```
logging.basicConfig(level=logging.DEBUG, format = ' ')
```

程序实例 ch15_24.py：重新设计 ch15_22.py，取消显示 logging 的前导输出消息。

```
1  # ch15_24.py
2  import logging
3
4  logging.basicConfig(level=logging.DEBUG, format='')
5  logging.debug('logging message, DEBUG')
6  logging.info('logging message, INFO')
7  logging.warning('logging message, WARNING')
8  logging.error('logging message, ERROR')
9  logging.critical('logging message, CRITICAL')
```

执行结果

```
==================== RESTART: D:/Python/ch15/ch15_24.py ====================
logging message, DEBUG
logging message, INFO
logging message, WARNING
logging message, ERROR
logging message, CRITICAL
>>>
```

上述执行结果很明显，模式前导的输出消息没有了。

15-7-4　时间信息 asctime

我们可以在 format 内配合 asctime 列出系统时间，这样可以列出每一重要阶段关键变量发生的时间。

程序实例 ch15_25.py：列出每一个 logging 输出时的时间。

```
1  # ch15_25.py
2  import logging
3
4  logging.basicConfig(level=logging.DEBUG, format='%(asctime)s')
5  logging.debug('logging message, DEBUG')
6  logging.info('logging message, INFO')
7  logging.warning('logging message, WARNING')
8  logging.error('logging message, ERROR')
9  logging.critical('logging message, CRITICAL')
```

执行结果

```
==================== RESTART: D:\Python\ch15\ch15_25.py ====================
2017-10-07 00:46:15,030
2017-10-07 00:46:15,030
2017-10-07 00:46:15,046
2017-10-07 00:46:15,046
2017-10-07 00:46:15,046
>>>
```

我们的确获得了每一个 logging 的输出时间，但是经过 format 处理后，原先 logging.xxx( ) 内的输出信息却没有了，这是因为我们在 format 内只有保留时间字符串消息。

15-7-5　format 内的 message

如果想要输出原先 logging.xxx( ) 的输出消息，必须在 format 内增加 message。

程序实例 ch15_26.py：增加 logging.xxx( ) 的输出消息。

```
1  # ch15_26.py
2  import logging
3
4  logging.basicConfig(level=logging.DEBUG, format='%(asctime)s : %(message)s')
5  logging.debug('logging message, DEBUG')
6  logging.info('logging message, INFO')
7  logging.warning('logging message, WARNING')
8  logging.error('logging message, ERROR')
9  logging.critical('logging message, CRITICAL')
```

执行结果

```
==================== RESTART: D:/Python/ch15/ch15_26.py ====================
2017-10-07 00:55:47,378 : logging message, DEBUG
2017-10-07 00:55:47,378 : logging message, INFO
2017-10-07 00:55:47,394 : logging message, WARNING
2017-10-07 00:55:47,394 : logging message, ERROR
2017-10-07 00:55:47,394 : logging message, CRITICAL
>>>
```

15-7-6　列出 levelname

levelname 属性是记载目前 logging 的显示层级是哪一个等级。

程序实例 ch15_27.py：列出目前 level 所设置的等级。

```
1  # ch15_27.py
2  import logging
3
4  logging.basicConfig(level=logging.DEBUG,
5                      format='%(asctime)s - %(levelname)s : %(message)s')
6  logging.debug('logging message.')
7  logging.info('logging message.')
8  logging.warning('logging message')
9  logging.error('logging message')
10 logging.critical('logging message')
```

执行结果

```
==================== RESTART: D:/Python/ch15/ch15_27.py ====================
2017-10-07 01:07:23,543 - DEBUG : logging message.
2017-10-07 01:07:23,543 - INFO : logging message.
2017-10-07 01:07:23,558 - WARNING : logging message
2017-10-07 01:07:23,558 - ERROR : logging message
2017-10-07 01:07:23,558 - CRITICAL : logging message
>>>
```

15-7-7 使用 logging 列出变量变化的应用

这一节开始将正式使用 logging 追踪变量的变化，下面是简单追踪索引值变化的程序。

程序实例 ch15_28.py：追踪索引值变化的实例。

```
1  # ch15_28.py
2  import logging
3
4  logging.basicConfig(level=logging.DEBUG,
5                      format='%(asctime)s - %(levelname)s : %(message)s')
6  logging.debug('程序开始')
7  for i in range(5):
8      logging.debug('目前索引 %s ' % i)
9  logging.debug('程序结束')
```

执行结果

```
===================== RESTART: D:\Python\ch15\ch15_28.py =====================
2017-12-16 12:46:29,765 - DEBUG : 程序开始
2017-12-16 12:46:29,765 - DEBUG : 目前索引 0
2017-12-16 12:46:29,765 - DEBUG : 目前索引 1
2017-12-16 12:46:29,765 - DEBUG : 目前索引 2
2017-12-16 12:46:29,780 - DEBUG : 目前索引 3
2017-12-16 12:46:29,780 - DEBUG : 目前索引 4
2017-12-16 12:46:29,780 - DEBUG : 程序结束
>>>
```

上述程序记录了整个索引值的变化过程，读者需留意第 8 行的输出，它的输出结果是在 %（message）s 定义的。

15-7-8 正式追踪 factorial 数值的应用

在程序 ch11_26.py 中曾经使用递归函数计算阶乘 factorial，接下来笔者想用一般循环方式追踪阶乘计算的过程。

程序实例 ch15_29.py：使用 logging 追踪 factorial 阶乘计算的过程。

```
1  # ch15_29.py
2  import logging
3
4  logging.basicConfig(level=logging.DEBUG,
5                      format='%(asctime)s - %(levelname)s : %(message)s')
6  logging.debug('程序开始')
7
8  def factorial(n):
9      logging.debug('factorial %s 计算开始' % n)
10     ans = 1
11     for i in range(n + 1):
12         ans *= i
13         logging.debug('i = ' + str(i) + ', ans = ' + str(ans))
14     logging.debug('factorial %s 计算结束' % n)
15     return ans
16
17 num = 5
18 print("factorial(%d) = %d" % (num, factorial(num)))
19 logging.debug('程序结束')
```

执行结果

```
===================== RESTART: D:\Python\ch15\ch15_29.py =====================
2017-12-16 12:49:30,468 - DEBUG : 程序开始
2017-12-16 12:49:30,468 - DEBUG : factorial 5 计算开始
2017-12-16 12:49:30,468 - DEBUG : i = 0, ans = 0
2017-12-16 12:49:30,468 - DEBUG : i = 1, ans = 0
2017-12-16 12:49:30,484 - DEBUG : i = 2, ans = 0
2017-12-16 12:49:30,484 - DEBUG : i = 3, ans = 0
2017-12-16 12:49:30,484 - DEBUG : i = 4, ans = 0
2017-12-16 12:49:30,484 - DEBUG : i = 5, ans = 0
2017-12-16 12:49:30,484 - DEBUG : factorial 5 计算结束
factorial(5) = 0
2017-12-16 12:49:30,500 - DEBUG : 程序结束
>>>
```

在上述语句使用 logging 的 DEBUG 过程中可以发现阶乘数从 0 开始，造成所有阶段的执行结果都是 0。在下列程序第 11 行，笔者更改此项设置为从 1 开始。

程序实例 ch15_30.py：修改 ch15_29.py 的错误，让阶乘从 1 开始。

```
1   # ch15_30.py
2   import logging
3
4   logging.basicConfig(level=logging.DEBUG,
5                       format='%(asctime)s - %(levelname)s : %(message)s')
6   logging.debug('程序开始')
7
8   def factorial(n):
9       logging.debug('factorial %s 计算开始' % n)
10      ans = 1
11      for i in range(1, n + 1):
12          ans *= i
13          logging.debug('i = ' + str(i) + ', ans = ' + str(ans))
14      logging.debug('factorial %s 计算结束' % n)
15      return ans
16
17  num = 5
18  print("factorial(%d) = %d" % (num, factorial(num)))
19  logging.debug('程序结束')
```

执行结果

```
==================== RESTART: D:\Python\ch15\ch15_30.py ====================
2017-12-16 12:52:39,994 - DEBUG : 程序开始
2017-12-16 12:52:39,994 - DEBUG : factorial 5 计算开始
2017-12-16 12:52:39,994 - DEBUG : i = 1, ans = 1
2017-12-16 12:52:40,009 - DEBUG : i = 2, ans = 2
2017-12-16 12:52:40,009 - DEBUG : i = 3, ans = 6
2017-12-16 12:52:40,009 - DEBUG : i = 4, ans = 24
2017-12-16 12:52:40,009 - DEBUG : i = 5, ans = 120
2017-12-16 12:52:40,009 - DEBUG : factorial 5 计算结束
factorial(5) = 120
2017-12-16 12:52:40,025 - DEBUG : 程序结束
>>>
```

15-7-9　将程序日志 logging 输出到文件

程序很长时，若将 logging 输出到屏幕，其实不太方便逐一核对关键变量值的变化，此时可以考虑将 logging 输出到文件，方法是在 logging.basicConfig( ) 中增加 filename=" 文件名 "，这样就可以将 logging 输出到指定的文件内。

程序实例 ch15_31.py：将程序实例的 logging 输出到 out15_31.txt。

```
4   logging.basicConfig(filename='out15_31.txt', level=logging.DEBUG,
5                       format='%(asctime)s - %(levelname)s : %(message)s')
```

执行结果

```
==================== RESTART: D:/Python/ch15/ch15_31.py ====================
factorial(5) = 120
>>>
```

这时在目前工作文件夹可以看到 out15_31.txt，打开后可以得到下列结果。

```
2019-06-03 00:27:59,482 - DEBUG : Program start
2019-06-03 00:27:59,483 - DEBUG : factorial 5 counting be
2019-06-03 00:27:59,483 - DEBUG : i = 1, ans = 1
2019-06-03 00:27:59,483 - DEBUG : i = 2, ans = 2
2019-06-03 00:27:59,484 - DEBUG : i = 3, ans = 6
2019-06-03 00:27:59,484 - DEBUG : i = 4, ans = 24
2019-06-03 00:27:59,484 - DEBUG : i = 5, ans = 120
2019-06-03 00:27:59,484 - DEBUG : factorial 5 end of coun
2019-06-03 00:27:59,497 - DEBUG : End of Program
```

15-7-10 隐藏程序日志 logging 的 DEBUG 等级使用 CRITICAL

先前有说明 logging 有许多等级，只要设置高等级，Python 就会忽略低等级的输出，所以如果程序设计完成，也确定没有错误，其实可以将 logging 等级设为最高等级，所有较低等级的输出将被隐藏。

程序实例 ch15_32.py：重新设计 ch15_30.py，将程序内 DEBUG 等级的 logging 隐藏。

```
4  logging.basicConfig(level=logging.CRITICAL,
5                      format='%(asctime)s - %(levelname)s : %(message)s')
```

执行结果

```
===================== RESTART: D:/Python/ch15/ch15_32.py =====================
factorial(5) = 120
>>>
```

15-7-11 停用程序日志 logging

可以使用下列方法停用日志 logging。

```
logging.disable(level)          # level 是停用 logging 的等级
```

上述语句可以停用该程序代码后指定等级以下的所有等级，如果想停用全部参数可以使用 logging.CRITICAL 等级，这个方法一般是放在 import 下方，这样就可以停用所有的 logging。

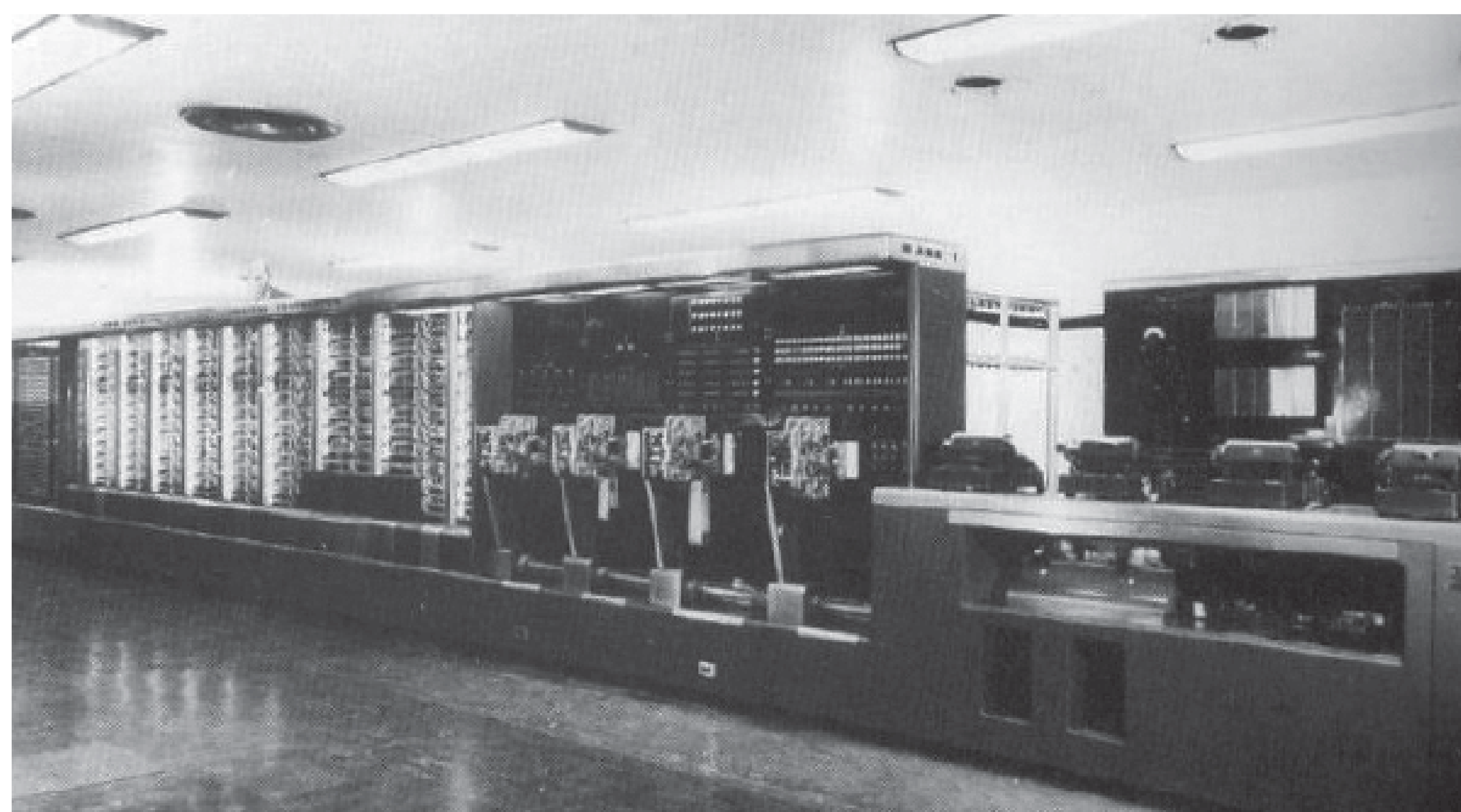
程序实例 ch15_33.py：重新设计 ch15_30.py，这个程序只是在原先第 3 行空白行加上下列程序代码。

```
3  logging.disable(logging.CRITICAL)        # 停用所有logging
```

执行结果 与 ch15_32.py 相同。

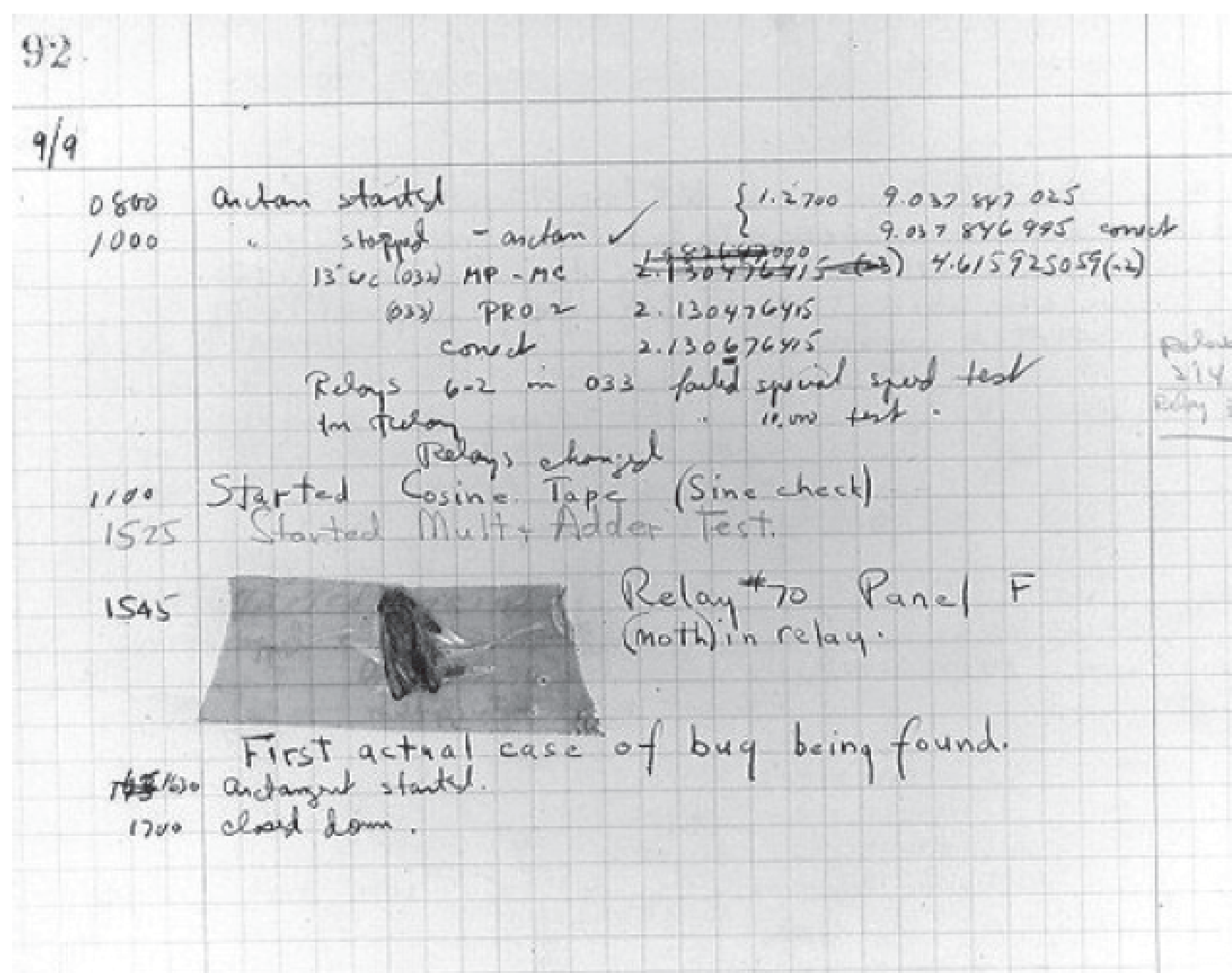
15-8 程序除错的典故

通常又将程序除错称为 Debug，De 是除去的意思，bug 是指小虫，其实这是有典故的。1944 年，IBM 和哈佛大学联合开发了 Mark I 计算机，此计算机重 5 吨，约有 2.4 米高，15.5 米长，内部线路总长是 800 多米，没有中断地使用了 15 年，下列是此计算机图片。



本图片转载自 http://www.computersciencelab.com

在当时有一位女性程序设计师 Grace Hopper，发现了第一个计算机虫（bug）——一只死的蛾（moth）的双翅卡在继电器（relay）中，导致数据读取失败。下列是当时 Grace Hopper 记录此事件的数据。



本图片转载自 http://www.computersciencelab.com

当时 Grace Hopper 写下了下列两句话。

```
Relay #70 Panel F (moth) in relay.
First actual case of bug being found.
```

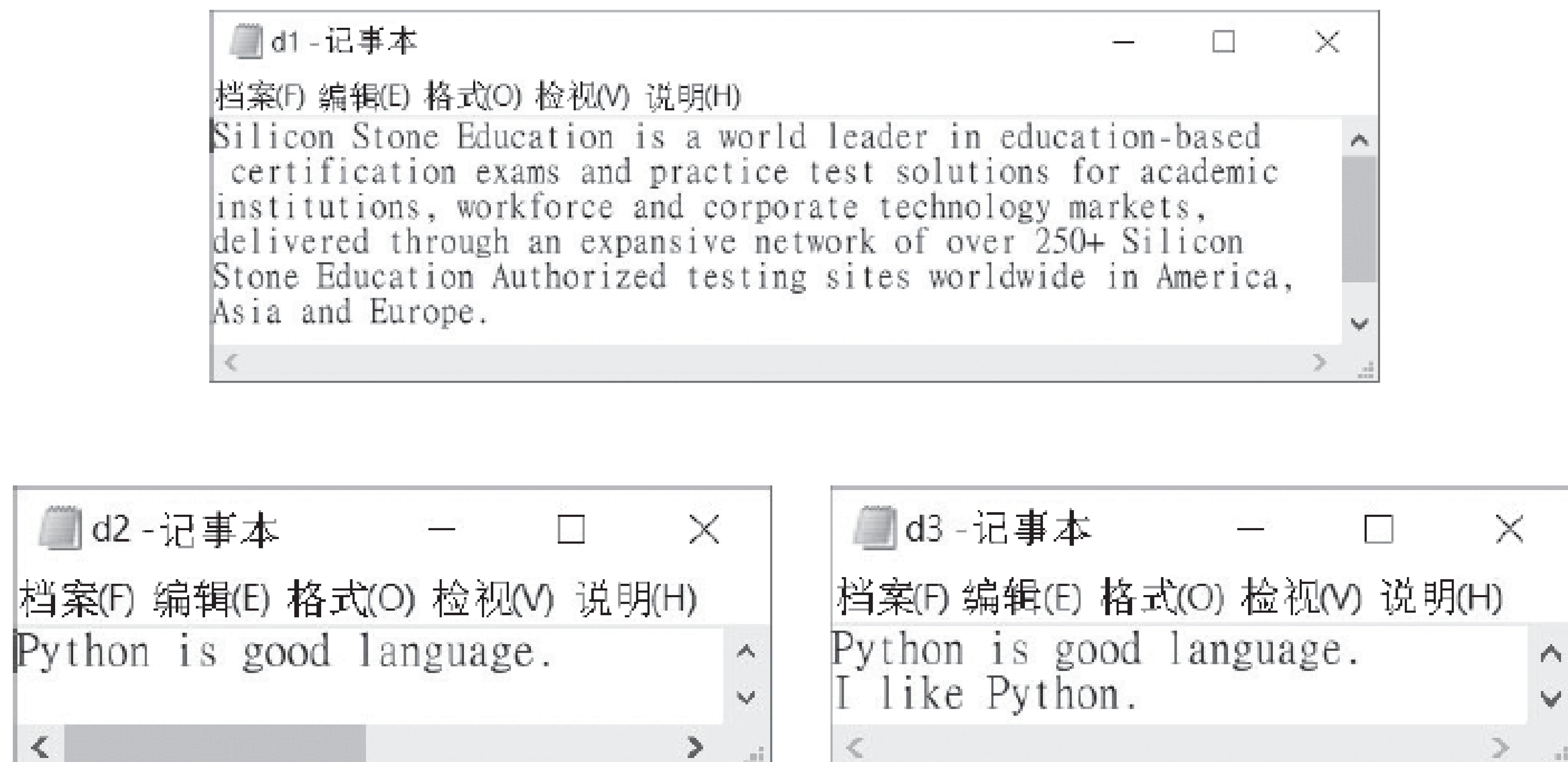
大意是编号 70 的继电器出现问题（因为蛾），这是真实计算机上所发现的第一只虫。自此，计算机界认定用 debug 描述找出及删除程序错误应归功于 Grace Hopper。

习题

1. 请将程序实例 ch15_6.py 改为由屏幕输入文字，然后将输入的文字存入 in15_6.txt，再予以分析。(15-1 节)

```
======================= RESTART: D:\Python\ex\ex15_1.py =======================
请输入文字 : I like Python. Python is a good language.
in15_6.txt  文章的字数是  8
```

2. 请将程序实例 ch15_8.py 第 13 行的 3 个文件改为 5 个文件，同时这 5 个文件的文件名（d1.txt, d2.txt, d3.txt, d4.txt, d5.txt）是由屏幕输入，内容如下。(15-1 节)


d1 - 记事本

档案(F) 编辑(E) 格式(O) 检视(V) 说明(H)

Silicon Stone Education is a world leader in education-based certification exams and practice test solutions for academic institutions, workforce and corporate technology markets, delivered through an expansive network of over 250+ Silicon Stone Education Authorized testing sites worldwide in America, Asia and Europe.

d2 - 记事本

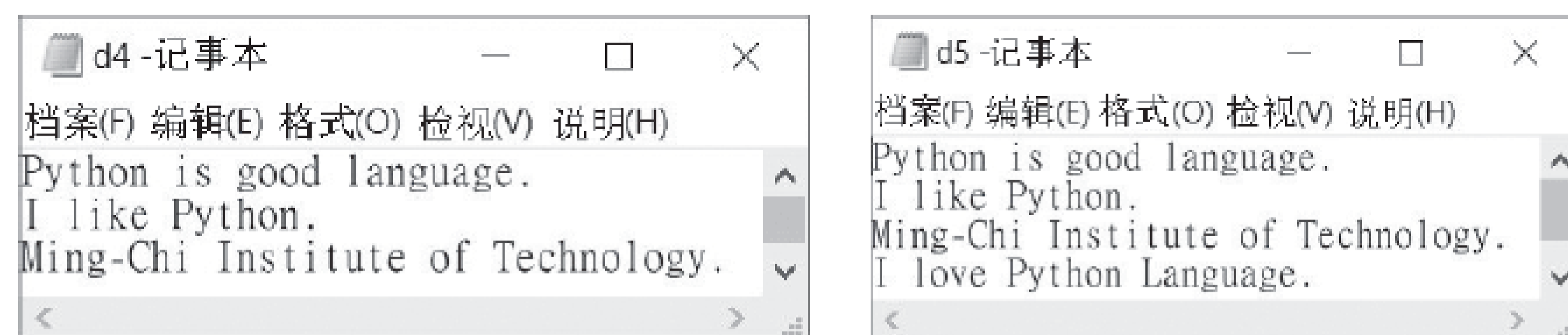
档案(F) 编辑(E) 格式(O) 检视(V) 说明(H)

Python is good language.

d3 - 记事本

档案(F) 编辑(E) 格式(O) 检视(V) 说明(H)

Python is good language.
I like Python.




```
===================== RESTART: D:\Python\ex\ex15_2.py =====================
请输入文件名 : d1.txt
请输入文件名 : d2.txt
请输入文件名 : d3.txt
请输入文件名 : d4.txt
请输入文件名 : d5.txt
d1.txt  文章的字数是  43
d2.txt  文章的字数是  4
d3.txt  文章的字数是  7
d4.txt  文章的字数是  11
d5.txt  文章的字数是  15
```

3. 请重新设计 ch15_11.py，但是将除数与被除数改为由屏幕输入。提示：使用 input( ) 读取输入时，所读取的是字符串，需使用 int( ) 将字符串转为整数数据类型，如果所输入的是非数字将产生 ValueError。（15-2 节）

```
===================== RESTART: D:\Python\ex\ex15_3.py =====================
请输入第1个数字 : 10
请输入第2个数字 : 2
5.0
>>>
===================== RESTART: D:\Python\ex\ex15_3.py =====================
请输入第1个数字 : 10
请输入第2个数字 : a
除法数据型态不符
None
>>>
===================== RESTART: D:\Python\ex\ex15_3.py =====================
请输入第1个数字 : 10
请输入第2个数字 : 0
除数不可为0
None
```

4. 请重新设计习题 3，但是只能有一个 except，可以捕捉所有错误，捕捉到错误时一律输出“数据输入错误”。（15-2 节）

```
===================== RESTART: D:\Python\ex\ex15_4.py =====================
请输入第1个数字 : 10
请输入第2个数字 : 2
5.0
>>>
===================== RESTART: D:\Python\ex\ex15_4.py =====================
请输入第1个数字 : 10
请输入第2个数字 : a
数据输入错误
None
>>>
===================== RESTART: D:\Python\ex\ex15_4.py =====================
请输入第1个数字 : 10
请输入第2个数字 : 0
数据输入错误
None
```

5. 请重新设计 ex15_4.py，以无限循环方式读取数据，如果输入‘q’或‘Q’代表程序结束。（15-2 节）

```
===================== RESTART: D:\Python\ex\ex15_5.py =====================
请输入第1个数字 : 10
请输入第2个数字 : 2
5.0
是否继续(y/n)，输入n或N代表不继续 ? y
请输入第1个数字 : 10
请输入第2个数字 : a
数据输入错误
None
是否继续(y/n)，输入n或N代表不继续 ? n
```

6. 请重新设计程序实例 ch15_15.py，将程序改为读取文件，请使用 ex15_2.py 的 5 个文件测试，如果文件长度超过 35 个字或小于 10 个字则出现异常。(15-3 节)

```
====================== RESTART: D:\Python\ex\ex15_6.py ======================
d1.txt  文章的字数是  43
文件长度检查异常发生:  文件长度太长
d2.txt  文章的字数是  4
文件长度检查异常发生:  文件长度不足
d3.txt  文章的字数是  7
文件长度检查异常发生:  文件长度不足
d4.txt  文章的字数是  11
文件长度正确
d5.txt  文章的字数是  15
文件长度正确
```

7. 请重新设计 ex15_6.py，当异常发生时，请将异常结果存入 errdata.txt 内，列出执行结果，同时列出 errdata.txt。(15-4 节)

```
====================== RESTART: D:\Python\ex\ex15_7.py ======================
d1.txt  文章的字数是  43
将Traceback写入错误文件errdata.txt完成
文件长度检查异常发生:  文件长度太长
d2.txt  文章的字数是  4
将Traceback写入错误文件errdata.txt完成
文件长度检查异常发生:  文件长度不足
d3.txt  文章的字数是  7
将Traceback写入错误文件errdata.txt完成
文件长度检查异常发生:  文件长度不足
d4.txt  文章的字数是  11
文件长度正确
d5.txt  文章的字数是  15
文件长度正确
```

下列是 errdata.txt 文件。

```
Traceback (most recent call last):
  File "D:/Python/ex/ex15_7.py", line 27, in <module>
    lenWord(file)
  File "D:/Python/ex/ex15_7.py", line 22, in lenWord
    raise Exception('文件长度太长')
Exception: 文件长度太长
Traceback (most recent call last):
  File "D:/Python/ex/ex15_7.py", line 27, in <module>
    lenWord(file)
  File "D:/Python/ex/ex15_7.py", line 20, in lenWord
    raise Exception('文件长度不足')
Exception: 文件长度不足
Traceback (most recent call last):
  File "D:/Python/ex/ex15_7.py", line 27, in <module>
    lenWord(file)
  File "D:/Python/ex/ex15_7.py", line 20, in lenWord
    raise Exception('文件长度不足')
Exception: 文件长度不足
```

8. 请重新设计 ch15_20.py，增加 __init__( )，需具有确定开户时金额在 100 元（含）以上的断言 assert。原程序第 27、28 行改为类似 23、24 行，但是使用新的变量名称。(15-7 节)

```
====================== RESTART: D:\Python\ex\ex15_8.py ======================
Hung  目前余额:  100
存款  300  完成
Hung  目前余额:  400
Traceback (most recent call last):
  File "D:\Python\ex\ex15_8.py", line 28, in <module>
    chenbank = Banks('chen', -100)
  File "D:\Python\ex\ex15_8.py", line 7, in __init__
    assert money >= 100, '开户金额必须大于或等于100'
AssertionError: 开户金额必须大于或等于100
```

9. 请参考程序实例 ch15_30.py，将 factorial(n) 函数改为 sumrange(n)，这个函数可以累计 1+2+…+n 的总和。(15-7 节)

```
====================== RESTART: D:\Python\ex\ex15_9.py ======================
2019-06-03 01:46:01,762 - DEBUG : 程序开始
2019-06-03 01:46:01,795 - DEBUG : sumrange 5 计算开始
2019-06-03 01:46:01,801 - DEBUG : i = 1, ans = 1
2019-06-03 01:46:01,810 - DEBUG : i = 2, ans = 3
2019-06-03 01:46:01,817 - DEBUG : i = 3, ans = 6
2019-06-03 01:46:01,823 - DEBUG : i = 4, ans = 10
2019-06-03 01:46:01,830 - DEBUG : i = 5, ans = 15
2019-06-03 01:46:01,837 - DEBUG : sumrange 5 计算结束
sumrange(5) = 15
2019-06-03 01:46:01,854 - DEBUG : 程序结束
```

16

第 16 章

正则表达式

本章摘要

正则表达式（Regular Expression）主要功能是执行模式的比对与查找，甚至 Word 文件也可以使用正则表达式处理**查找**（search）与**替换**（replace）功能。本章首先会介绍如果没用正则表达式，如何处理查找文字功能，再介绍使用正则表达式处理这类问题，读者会发现整个工作变得更简洁容易。

16-1 使用 Python 硬功夫查找文字

如果现在打开手机的联络信息可以看到，台湾地区手机号码的格式如下：

0952-282-020　　　　　　# 可以表示为 xxxx-xxx-xxx，每个 x 代表一个 0 ～ 9 数字

可以发现手机号码格式是 4 个数字，1 个连字符号，3 个数字，1 个连字符号，3 个数字所组成。

程序实例 ch16_1.py：用传统知识设计一个程序，然后判断字符串是否有含台湾地区的手机号码格式。

```
1  # ch16_1.py
2  def taiwanPhoneNum(string):
3      """检查是否有含手机联络信息的台湾地区手机号码格式"""
4      if len(string) != 12:          # 如果长度不是12
5          return False               # 返回非手机号码格式
6
7      for i in range(0, 4):          # 如果前4个字出现非数字字符
8          if string[i].isdecimal() == False:
9              return False           # 返回非手机号码格式
10
11     if string[4] != '-':           # 如果不是'-'字符
12         return False               # 返回非手机号码格式
13
14     for i in range(5, 8):          # 如果中间3个字出现非数字字符
15         if string[i].isdecimal() == False:
16             return False           # 返回非手机号码格式
17
18     if string[8] != '-':           # 如果不是'-'字符
19         return False               # 返回非手机号码格式
20
21     for i in range(9, 12):         # 如果最后3个字出现非数字字符
22         if string[i].isdecimal() == False:
23             return False           # 返回非手机号码格式
24     return True                    # 通过以上测试
25
26 print("I love Ming-Chi: 是台湾地区手机号码", taiwanPhoneNum('I love Ming-Chi'))
27 print("0932-999-199:    是台湾地区手机号码", taiwanPhoneNum('0932-999-199'))
```

执行结果

```
===================== RESTART: D:\Python\ch16\ch16_1.py =====================
I love Ming-Chi: 是台湾地区手机号码 False
0932-999-199:    是台湾地区手机号码 True
>>>
```

上述程序第 4 和 5 行是判断字符串长度是否为 12，如果不是则表示这不是手机号码格式。程序第 7 ～ 9 行是判断字符串前 4 个码是不是数字，如果不是则表示这不是手机号码格式，注：如果是数字字符 isdecimal( ) 会返回 True。程序第 11、12 行是判断这个字符是不是 ‘-’，如果不是则表示这不是手机号码格式。程序第 14 ～ 16 行是判断字符串索引 [5][6][7] 码是不是数字，如果不是则表示这不是手机号码格式。程序第 18、19 行是判断这个字符是不是 ‘-’，如果不是则表示这不是手机号码格式。程序第 21 ～ 23 行是判断字符串索引 [9][10][11] 码是不是数字，如果不是则表示这不是手机号码格式。如果通过了以上所有测试，表示这是手机号码格式，程序第 24 行返回 True。

在真实的环境应用中，我们可能需面临一段文字，这段文字内穿插一些数字，然后我们必须将手机号码从这段文字抽离出来。

程序实例 ch16_2.py：将电话号码从一段文字抽离出来。

```
1   # ch16_2.py
2   def taiwanPhoneNum(string):
3       """检查是否有含手机联络信息的台湾地区手机号码格式"""
4       if len(string) != 12:          # 如果长度不是12
5           return False               # 返回非手机号码格式
6
7       for i in range(0, 4):          # 如果前4个字出现非数字字符
8           if string[i].isdecimal() == False:
9               return False           # 返回非手机号码格式
10
11      if string[4] != '-':           # 如果不是'-'字符
12          return False               # 返回非手机号码格式
13
14      for i in range(5, 8):          # 如果中间3个字出现非数字字符
15          if string[i].isdecimal() == False:
16              return False           # 返回非手机号码格式
17
18      if string[8] != '-':           # 如果不是'-'字符
19          return False               # 返回非手机号码格式
20
21      for i in range(9, 12):         # 如果最后3个字出现非数字字符
22          if string[i].isdecimal() == False:
23              return False           # 返回非手机号码格式
24      return True                    # 通过以上测试
25
26  def parseString(string):
27      """解析字符串是否含有电话号码"""
28      notFoundSignal = True          # 注记没有找到电话号码为True
29      for i in range(len(string)):   # 用循环逐步抽取12个字符做测试
30          msg = string[i:i+12]
31          if taiwanPhoneNum(msg):
32              print("电话号码是: %s" % msg)
33              notFoundSignal = False
34      if notFoundSignal:             # 如果没有找到电话号码则打印
35          print("%s 字符串不含电话号码" % string)
36
37  msg1 = 'Please call my secretary using 0930-919-919 or 0952-001-001'
38  msg2 = '请明天17:30和我一起参加明志科大教师节晚餐'
39  msg3 = '请明天17:30和我一起参加明志科大教师节晚餐，可用0933-080-080联络我'
40  parseString(msg1)
41  parseString(msg2)
42  parseString(msg3)
```

执行结果

```
==================== RESTART: D:\Python\ch16\ch16_2.py ====================
电话号码是: 0930-919-919
电话号码是: 0952-001-001
请明天17:30和我一起参加明志科大教师节晚餐 字符串不含电话号码
电话号码是: 0933-080-080
>>>
```

从上述执行结果可以得到，我们成功地对一个字符串分析，然后将电话号码分析出来了。分析方式的重点是程序第 26 ～ 35 行的 parseString 函数，这个函数重点是第 29 ～ 33 行，这个循环会逐步抽取字符串的 12 个字符做比对，将比对字符串放在 msg 字符串变量内。下列是各循环次序的 msg 字符串变量内容。

```
msg = 'Please call '   # 第 1 次 [0:12]
msg = 'lease call m'   # 第 2 次 [1:13]
msg = 'ease call my'   # 第 3 次 [2:14]
…
```

```
msg = '0930-939-939'            # 第 31 次 [30:42]
...
msg = '0952-001-001'            # 第 48 次 [47:59]
```

程序第 28 行将没有找到电话号码 notFoundSignal 设为 True，如果找到电话号码，程序 33 行将 notFoundSignal 标识为 False，当 parseString( ) 函数执行完，notFoundSignal 仍是 True，表示没找到电话号码，所以第 35 行打印 **"字符串不含电话号码"**。

上述使用所学的 Python 硬功夫虽然解决了我们的问题，但是若是将电话号码改成中国大陆手机号（xxx-xxxx-xxxx）、美国手机号（xxx-xxx-xxxx）或是一般公司的电话，整个号码格式不一样，要重新设计可能需要一些时间。不过不用担心，接下来将讲解 Python 的正则表达式，可以轻松解决上述困扰。

16-2　正则表达式的基础

Python 有关正则表达式的方法是在 re 模块内，所以使用正则表达式需要导入 re 模块。

```
import  re                    # 导入 re 模块
```

16-2-1　建立查找字符串模式

在 16-1 节使用 isdecimal( ) 方法判断字符是否为 0 ～ 9 的数字。

正则表达式是一种文本模式的表达方法，在这个方法中，使用 \d 表示 0 ～ 9 的数字字符，采用这个概念可以将 16-1 节的手机号码 xxxx-xxx-xxx 改用下列正则表达方式表示。

```
'\d\d\d\d-\d\d\d-\d\d\d'
```

由转义字符的概念可知，将上述表达式当字符串放入函数内需增加 '\'，所以整个正则表达式的使用方式如下。

```
'\\d\\d\\d\\d-\\d\\d\\d-\\d\\d\\d'
```

在 3-4-9 节有介绍字符串前加 r 可以防止字符串内的转义字符被转义，所以又可以将上述正则表达式简化为下列格式。

```
r'\d\d\d\d-\d\d\d-\d\d\d'
```

16-2-2　使用 re.compile( ) 建立 Regex 对象

Regex 是 Regular expression 的简称，在 re 模块内有 compile( ) 方法，可以将 16-2-1 节的要查找字符串的正则表达式当作字符串参数放在此方法内，然后会返回一个 Regex 对象，如下所示。

```
phoneRule = re.compile(r'\d\d\d\d-\d\d\d-\d\d\d')  # 建立 phoneRule 对象
```

16-2-3　查找对象

在 Regex 对象内有 search( ) 方法，可以由 Regex 对象启用，然后将要查找的字符串放在这个方法内，沿用上述概念程序片段如下。

```
phoneNum = phoneRule.search(msg)                # msg 是要查找的字符串
```

如果找不到比对相符的字符串会返回 None，如果找到比对相符的字符串会将结果返回所设置的 phoneNum 变量对象，这个对象在 Python 中称为 MatchObject 对象，将在 16-6 节完整解说。现在将介绍实用性较高的部分，处理此对象主要是将查找结果返回，可以用 group( ) 方法将结果返回，不过 search( ) 将只返回第一个比对相符的字符串。

程序实例 ch16_3.py：使用正则表达式重新设计 ch16_2.py。

```
 1  # ch16_3.py
 2  import re
 3
 4  msg1 = 'Please call my secretary using 0930-919-919 or 0952-001-001'
 5  msg2 = '请明天17:30和我一起参加明志科大教师节晚餐'
 6  msg3 = '请明天17:30和我一起参加明志科大教师节晚餐，可用0933-080-080联络我'
 7
 8  def parseString(string):
 9      """解析字符串是否含有电话号码"""
10      phoneRule = re.compile(r'\d\d\d\d-\d\d\d-\d\d\d')
11      phoneNum = phoneRule.search(string)
12      if phoneNum != None:          # 检查phoneNum内容
13          print("电话号码是: %s" % phoneNum.group())
14      else:
15          print("%s 字符串不含电话号码" % string)
16
17  parseString(msg1)
18  parseString(msg2)
19  parseString(msg3)
```

执行结果

```
===================== RESTART: D:\Python\ch16\ch16_3.py =====================
电话号码是: 0930-919-919
请明天17:30和我一起参加明志科大教师节晚餐 字符串不含电话号码
电话号码是: 0933-080-080
>>>
```

在程序实例 ch16_2.py 中使用了约 21 行做字符串解析，当我们使用 Python 的正则表达式时，只用第 10 和 11 行共两行就解析了字符串是否含手机号码了，整个程序变得简单许多。不过上述 msg1 字符串内含两组手机号码，使用 search( ) 只返回第一个发现的号码，16-2-4 节将改进此方法。

16-2-4 findall( )

从方法的名字就可以知道，这个方法可以返回所有找到的手机号码。这个方法会将查找到的手机号码用列表方式返回，这样就不会有只显示第一个查找到手机号码的缺点。如果没有比对相符的号码就返回 [ ] 空列表。要使用这个方法的关键指令如下。

```
phoneRule = re.compile(r'\d\d\d\d-\d\d\d-\d\d\d')   # 建立 phoneRule 对象
    phoneNum = phoneRule.findall(string)          # string 是要查找的字符串
```

findall( ) 函数由 phoneRule 对象启用，最后会将查找结果的列表传给 phoneNum，只要打印 phoneNum 就可以得到执行结果。

程序实例 ch16_4.py：使用 findall( ) 查找字符串，第 10 行定义正则表达式，程序会打印结果。

```
1   # ch16_4.py
2   import re
3
4   msg1 = 'Please call my secretary using 0930-919-919 or 0952-001-001'
5   msg2 = '请明天17:30和我一起参加明志科大教师节晚餐'
6   msg3 = '请明天17:30和我一起参加明志科大教师节晚餐，可用0933-080-080联络我'
7
8   def parseString(string):
9       """解析字符串是否含有电话号码"""
10      phoneRule = re.compile(r'\d\d\d\d-\d\d\d-\d\d\d')
11      phoneNum = phoneRule.findall(string)      # 用列表返回查找结果
12      print("电话号码是: %s" % phoneNum)         # 列表方式显示电话号码
13
14  parseString(msg1)
15  parseString(msg2)
16  parseString(msg3)
```

执行结果

```
===================== RESTART: D:\Python\ch16\ch16_4.py =====================
电话号码是: ['0930-919-919', '0952-001-001']
电话号码是: []
电话号码是: ['0933-080-080']
>>>
```

16-2-5　再看 re 模块

其实 Python 语言的 re 模块对于 search( ) 和 findall( ) 有提供更强的功能，可以省略使用 re.compile( ) 直接将比对模式放在各自的参数内，语法格式如下。

```
re.search(pattern, string, flags)
re.findall(pattern, string, flags)
```

上述 pattern 是要查找的正则表达方式，string 是所查找的字符串，flags 可以省略，未来会介绍几个 flags 常用相关参数的应用。

程序实例 ch16_5.py：使用 re.search( ) 重新设计 ch16_3.py，由于省略了 re.compile( )，所以读者需留意第 11 行内容的写法。

```
1   # ch16_5.py
2   import re
3
4   msg1 = 'Please call my secretary using 0930-919-919 or 0952-001-001'
5   msg2 = '请明天17:30和我一起参加明志科大教师节晚餐'
6   msg3 = '请明天17:30和我一起参加明志科大教师节晚餐，可用0933-080-080联络我'
7
8   def parseString(string):
9       """解析字符串是否含有电话号码"""
10      pattern = r'\d\d\d\d-\d\d\d-\d\d\d'
11      phoneNum = re.search(pattern, string)
12      if phoneNum != None:          # 如果phoneNum不是None表示取得号码
13          print("电话号码是: %s" % phoneNum.group())
14      else:
15          print("%s 字符串不含电话号码" % string)
16
17  parseString(msg1)
18  parseString(msg2)
19  parseString(msg3)
```

执行结果　与 ch16_3.py 相同。

程序实例 ch16_6.py：使用 re.findall( ) 重新设计 ch16_4.py，由于省略了 re.compile( )，所以读者需留

意第 11 行内容的写法。

```
1   # ch16_6.py
2   import re
3
4   msg1 = 'Please call my secretary using 0930-919-919 or 0952-001-001'
5   msg2 = '请明天17:30和我一起参加明志科大教师节晚餐'
6   msg3 = '请明天17:30和我一起参加明志科大教师节晚餐，可用0933-080-080联络我'
7
8   def parseString(string):
9       """解析字符串是否含有电话号码"""
10      pattern = r'\d\d\d\d-\d\d\d-\d\d\d'
11      phoneNum = re.findall(pattern, string)   # 用列表返回查找结果
12      print("电话号码是: %s" % phoneNum)        # 列表方式显示电话号码
13
14  parseString(msg1)
15  parseString(msg2)
16  parseString(msg3)
```

执行结果 与 ch16_4.py 相同。

16-2-6 再看正则表达式

下面是我们目前的正则表达式所查找的字符串模式。

r'\d\d\d\d-\d\d\d-\d\d\d'

其中可以看到 \d 重复出现，对于重复出现的字符串可以用大括号内部加上重复次数的方式表达，所以上述可以用下列方式表达。

r'\d{4}-\d{3}-\d{3}'

程序实例 ch16_7.py：使用本节概念重新设计 ch16_6.py，下面只列出不一样的程序内容。

```
10      pattern = r'\d{4}-\d{3}-\d{3}'
```

执行结果 与 ch16_4.py 相同。

16-3 更多查找比对模式

先前所用的实例是手机号码，试想想看如果改用市区电话号码的比对，中国台北市的电话号码如下。

02-28350000　　　　　　# 可用 xx-xxxxxxxx 表达

下面将以上述电话号码模式说明。

16-3-1 使用小括号分组

依照 16-2 节的概念，可以用下列正则表示法表达上述市区电话号码。

r'\d\d-\d\d\d\d\d\d\d\d'

所谓括号分组是以连字符“-”区别，然后用小括号隔开群组，可以用下列方式重新规划上述表达式。

r'(\d\d)-(\d\d\d\d\d\d\d\d')

也可简化为：

r'(\d{2})-(\d{8})'

当使用 re.search( ) 执行比对时，未来可以使用 group( ) 返回比对符合的不同分组，例如，group( ) 或 group(0) 返回第一个比对相符的文字，与 ch16_3.py 概念相同。group(1) 则返回括号的第一组文字，group(2) 则返回括号的第二组文字。

程序实例 ch16_8.py：使用小括号分组的概念，将个分组内容输出。

```
1  # ch16_8.py
2  import re
3
4  msg = 'Please call my secretary using 02-26669999'
5  pattern = r'(\d{2})-(\d{8})'
6  phoneNum = re.search(pattern, msg)         # 返回查找结果
7
8  print("完整号码是: %s" % phoneNum.group())  # 显示完整号码
9  print("完整号码是: %s" % phoneNum.group(0)) # 显示完整号码
10 print("区域号码是: %s" % phoneNum.group(1)) # 显示区域号码
11 print("电话号码是: %s" % phoneNum.group(2)) # 显示电话号码
```

执行结果

```
==================== RESTART: D:\Python\ch16\ch16_8.py ====================
完整号码是: 02-26669999
完整号码是: 02-26669999
区域号码是: 02
电话号码是: 26669999
>>>
```

如果所查找比对的正则表达式字符串有用小括号分组时，若是使用 findall( ) 方法处理，会返回元组（tuple）的列表（list），元组内的每个元素就是查找的分组内容。

程序实例 ch16_9.py：使用 findall( ) 重新设计 ch16_8.py，这个实例会多增加一组电话号码。

```
1  # ch16_9.py
2  import re
3
4  msg = 'Please call my secretary using 02-26669999 or 02-11112222'
5  pattern = r'(\d{2})-(\d{8})'
6  phoneNum = re.findall(pattern, msg)          # 返回查找结果
7  print(phoneNum)
```

执行结果

```
===================== RESTART: D:\Python\ch16\ch16_9.py =====================
[('02', '26669999'), ('02', '11112222')]
```

16-3-2　groups( )

注意这是 groups( )，在 group 后面加上了 s，当我们使用 re.search( ) 查找字符串时，可以使用这个方法取得分组的内容。这时还可以使用 2-9 节的多重指定的概念，例如，若以 ch16_8.py 为例，在第 7 行可以使用下列多重指定获得区域号码和当地电话号码。

```
areaNum, localNum = phoneNum.groups( )          # 多重指定
```

程序实例 ch16_10.py：重新设计 ch16_8.py，分别列出区域号码与电话号码。

```
1  # ch16_10.py
2  import re
3
4  msg = 'Please call my secretary using 02-26669999'
5  pattern = r'(\d{2})-(\d{8})'
6  phoneNum = re.search(pattern, msg)      # 返回查找结果
7  areaNum, localNum = phoneNum.groups()   # 留意是groups()
8  print("区域号码是: %s" % areaNum)        # 显示区域号码
9  print("电话号码是: %s" % localNum)       # 显示电话号码
```

执行结果

```
==================== RESTART: D:\Python\ch16\ch16_10.py ====================
区域号码是: 02
电话号码是: 26669999
>>>
```

16-3-3 区域号码是在小括号内

在一般电话号码的使用中，常看到区域号码是用小括号括起来，如下所示。

(02)-26669999

在处理小括号时，方式是 \（和 \），可参考下列实例。

程序实例 ch16_11.py：重新设计 ch16_10.py，第 4 行的区域号码是（02），读者需留意第 4 行和第 5 行的设计。

```
1  # ch16_11.py
2  import re
3
4  msg = 'Please call my secretary using (02)-26669999'
5  pattern = r'(\(\d{2}\))-(\d{8})'
6  phoneNum = re.search(pattern, msg)      # 返回查找结果
7  areaNum, localNum = phoneNum.groups()   # 留意是groups()
8  print("区域号码是: %s" % areaNum)        # 显示区域号码
9  print("电话号码是: %s" % localNum)       # 显示电话号码
```

执行结果

```
==================== RESTART: D:\Python\ch16\ch16_11.py ====================
区域号码是: (02)
电话号码是: 26669999
>>>
```

16-3-4 使用管道 |

|（pipe）在正则表示法中称为管道，使用管道时可以同时查找比对多个字符串，例如，如果想要查找 Mary 和 Tom 字符串，可以使用下列表示方法。

```
pattern = 'Mary|Tom'          # 注意单引号 ' 或 | 旁不可留空白
```

程序实例 ch16_12.py：管道查找多个字符串的实例。

```
1  # ch16_12.py
2  import re
3
4  msg = 'John and Tom will attend my party tonight. John is my best friend.'
5  pattern = 'John|Tom'                  # 查找John和Tom
6  txt = re.findall(pattern, msg)        # 返回查找结果
7  print(txt)
8  pattern = 'Mary|Tom'                  # 查找Mary和Tom
9  txt = re.findall(pattern, msg)        # 返回查找结果
10 print(txt)
```

执行结果

```
===================== RESTART: D:\Python\ch16\ch16_12.py =====================
['John', 'Tom', 'John']
['Tom']
```

16-3-5　多个分组的管道查找

假设有一个字符串内容如下：

Johnson, Johnnason and Johnnathan will attend my party tonight.

由上述可知如果想要查找字符串比对 John 后面可以是 son、nason、nathan 任一个字符串的组合，可以使用下列正则表达式格式。

pattern = 'John(son|nason|nathan)'

程序实例 ch16_13.py：查找 Johnson、Johnnason 或 Johnnathan 任一字符串，然后列出结果，这个程序将列出第一个查找比对到的字符串。

```
1  # ch16_13.py
2  import re
3
4  msg = 'Johnson, Johnnason and Johnnathan will attend my party tonight.'
5  pattern = 'John(son|nason|nathan)'
6  txt = re.search(pattern,msg)    # 返回查找结果
7  print(txt.group())              # 打印第一个查找结果
8  print(txt.group(1))             # 打印第一个分组
```

执行结果

```
===================== RESTART: D:\Python\ch16\ch16_13.py =====================
Johnson
son
```

同样的正则表达式若是使用 findall( ) 方法处理，将只返回各分组查找到的字符串，如果要列出完整的内容，可以用循环同时为每个分组字符串加上前导字符串 John。

程序实例 ch16_14.py：使用 findall( ) 重新设计 ch16_13.py。

```
1  # ch16_14.py
2  import re
3
4  msg = 'Johnson, Johnnason and Johnnathan will attend my party tonight.'
5  pattern = 'John(son|nason|nathan)'
6  txts = re.findall(pattern,msg)      # 返回查找结果
7  print(txts)
8  for txt in txts:                    # 将查找到内容加上John
9      print('John'+txt)
```

执行结果

```
===================== RESTART: D:\Python\ch16\ch16_14.py =====================
['son', 'nason', 'nathan']
Johnson
Johnnason
Johnnathan
```

16-3-6 使用 ? 做查找

在正则表达式中若是某些括号内的字符串或正则表达式可有可无，执行查找时都算成功，例如，na 字符串可有可无，表达方式是（na）?。

程序实例 ch16_15.py：使用 ? 查找的实例，这个程序会测试两次。

```
1  # ch16_15.py
2  import re
3  # 测试1
4  msg = 'Johnson will attend my party tonight.'
5  pattern = 'John((na)?son)'
6  txt = re.search(pattern,msg)       # 返回查找结果
7  print(txt.group())
8  # 测试2
9  msg = 'Johnnason will attend my party tonight.'
10 pattern = 'John((na)?son)'
11 txt = re.search(pattern,msg)       # 返回查找结果
12 print(txt.group())
```

执行结果

```
==================== RESTART: D:\Python\ch16\ch16_15.py ====================
Johnson
Johnnason
```

有时候如果居住在同一个城市，在留电话号码时，可能不会留区域号码，这时就可以使用本功能了。请参考下列实例第 11 行。

程序实例 ch16_16.py：这个程序在查找电话号码时，如果省略区域号码程序也可以查找到此号码，然后打印出来，正则表达式格式请留意第 6 行。

```
1  # ch16_16.py
2  import re
3
4  # 测试1
5  msg = 'Please call my secretary using 02-26669999'
6  pattern = r'(\d\d-)?(\d{8})'                 # 增加?号
7  phoneNum = re.search(pattern, msg)           # 返回查找结果
8  print("完整号码是: %s" % phoneNum.group())    # 显示完整号码
9
10 # 测试2
11 msg = 'Please call my secretary using 26669999'
12 pattern = r'(\d\d-)?(\d{8})'                 # 增加?号
13 phoneNum = re.search(pattern, msg)           # 返回查找结果
14 print("完整号码是: %s" % phoneNum.group())    # 显示完整号码
```

执行结果

```
==================== RESTART: D:\Python\ch16\ch16_16.py ====================
完整号码是: 02-26669999
完整号码是: 26669999
>>>
```

16-3-7 使用 * 号做查找

在正则表达式中若是某些字符串或正则表达式可为 0 到多次，执行查找时都算成功，例如，na

字符串可为 0 到多次，表达方式是（na）*。

程序实例 ch16_17.py：这个程序的重点是第 5 行的正则表达式，其中，字符串 na 的出现次数可以是 0 次到多次。

```
1   # ch16_17.py
2   import re
3   # 测试1
4   msg = 'Johnson will attend my party tonight.'
5   pattern = 'John((na)*son)'          # 字符串na可以为0到多次
6   txt = re.search(pattern,msg)        # 返回查找结果
7   print(txt.group())
8   # 测试2
9   msg = 'Johnnason will attend my party tonight.'
10  pattern = 'John((na)*son)'          # 字符串na可以为0到多次
11  txt = re.search(pattern,msg)        # 返回查找结果
12  print(txt.group())
13  # 测试3
14  msg = 'Johnnananason will attend my party tonight.'
15  pattern = 'John((na)*son)'          # 字符串na可以为0到多次
16  txt = re.search(pattern,msg)        # 返回查找结果
17  print(txt.group())
```

执行结果

```
===================== RESTART: D:\Python\ch16\ch16_17.py =====================
Johnson
Johnnason
Johnnananason
```

16-3-8　使用 + 号做查找

在正则表达式中若是某些字符串或正则表达式可为 1 到多次，执行查找时都算成功，例如，na 字符串可为 1 到多次，表达方式是（na）+。

程序实例 ch16_18.py：这个程序的重点是第 5 行的正则表达式，其中，字符串 na 的出现次数可以是 1 次到多次。

```
1   # ch16_18.py
2   import re
3   # 测试1
4   msg = 'Johnson will attend my party tonight.'
5   pattern = 'John((na)+son)'          # 字符串na可以为1到多次
6   txt = re.search(pattern,msg)        # 返回查找结果
7   print(txt)                          # 请注意是直接打印对象
8   # 测试2
9   msg = 'Johnnason will attend my party tonight.'
10  pattern = 'John((na)+son)'          # 字符串na可以为1到多次
11  txt = re.search(pattern,msg)        # 返回查找结果
12  print(txt.group())
13  # 测试3
14  msg = 'Johnnananason will attend my party tonight.'
15  pattern = 'John((na)+son)'          # 字符串na可以为1到多次
16  txt = re.search(pattern,msg)        # 返回查找结果
17  print(txt.group())
```

执行结果

```
===================== RESTART: D:\Python\ch16\ch16_18.py =====================
None
Johnnason
Johnnananason
```

16-3-9 查找时忽略大小写

查找时若是在 search( ) 或 findall( ) 内增加第三个参数 re.I 或 re.IGNORECASE，查找时就会忽略大小写，至于打印输出时将以原字符串的格式显示。

程序实例 ch16_19.py：以忽略大小写方式执行查找相符字符串。

```
1  # ch16_19.py
2  import re
3
4  msg = 'john and TOM will attend my party tonight. JOHN is my best friend.'
5  pattern = 'John|Tom'                        # 查找John和Tom
6  txt = re.findall(pattern, msg, re.I)        # 返回查找忽略大小写的结果
7  print(txt)
8  pattern = 'Mary|tom'                        # 查找Mary和tom
9  txt = re.findall(pattern, msg, re.I)        # 返回查找忽略大小写的结果
10 print(txt)
```

执行结果

```
===================== RESTART: D:\Python\ch16\ch16_19.py =====================
['john', 'TOM', 'JOHN']
['TOM']
```

16-4 贪婪与非贪婪查找

16-4-1 查找时使用大括号设置比对次数

在 16-2-6 节有使用过大括号，当时讲解 \d{4} 代表重复 4 次，也就是大括号中的数字是设置重复次数。可以将这个概念应用在查找一般字符串，例如，(son) {3} 代表所查找的字符串是 'sonsonson'，如果有一字符串是 'sonson'，则查找结果是不符。大括号除了可以设置重复次数，也可以设置指定范围，例如，(son) {3,5} 代表所查找的字符串如果是 'sonsonson' 'sonsonsonson' 或 'sonsonsonsonson' 都算是相符合的字符串。(son) {3,5} 正则表达式相当于下列表达式：

((son)(son)(son))|((son)(son)(son)(son))|((son)(son)(son)(son)(son))

程序实例 ch16_20.py：设置查找 son 字符串重复 3 ～ 5 次都算查找成功。

```
1  # ch16_20.py
2  import re
3
4  def searchStr(pattern, msg):
5      txt = re.search(pattern, msg)
6      if txt == None:              # 查找失败
7          print("查找失败 ",txt)
8      else:                        # 查找成功
9          print("查找成功 ",txt.group())
10
11 msg1 = 'son'
12 msg2 = 'sonson'
13 msg3 = 'sonsonson'
14 msg4 = 'sonsonsonson'
15 msg5 = 'sonsonsonsonson'
16 pattern = '(son){3,5}'
17 searchStr(pattern,msg1)
18 searchStr(pattern,msg2)
19 searchStr(pattern,msg3)
20 searchStr(pattern,msg4)
21 searchStr(pattern,msg5)
```

执行结果

```
===================== RESTART: D:\Python\ch16\ch16_20.py =====================
查找失败  None
查找失败  None
查找成功  sonsonson
查找成功  sonsonsonson
查找成功  sonsonsonsonson
```

使用大括号时，也可以省略第一个或第二个数字，这相当于不设置最小或最大重复次数。例如，（son）{3,} 代表重复 3 次以上都符合，（son）{,10} 代表重复 10 次以下都符合。有关这方面的实践，将留给读者练习，可参考习题 3。

16-4-2　贪婪与非贪婪查找

在讲解贪婪与非贪婪查找前，笔者先简化程序实例 ch16_20.py，使用相同的查找模式 '（son）{3,5}'，查找字符串 'sonsonsonsonson'，看看结果。

程序实例 ch16_21.py：使用查找模式 '（son）{3,5}' 查找字符串 'sonsonsonsonson'。

```
1  # ch16_21.py
2  import re
3
4  def searchStr(pattern, msg):
5      txt = re.search(pattern, msg)
6      if txt == None:           # 查找失败
7          print("查找失败 ",txt)
8      else:                     # 查找成功
9          print("查找成功 ",txt.group())
10
11 msg = 'sonsonsonsonson'
12 pattern = '(son){3,5}'
13 searchStr(pattern,msg)
```

执行结果

```
===================== RESTART: D:\Python\ch16\ch16_21.py =====================
查找成功  sonsonsonsonson
```

其实由上述程序所设置的查找模式可知，3、4 或 5 个 son 重复就算找到了，可是 Python 执行结果是列出重复最多的字符串，5 次重复，这是 Python 的默认模式，这种模式又称为贪婪（greedy）模式。

另一种是列出重复最少的字符串，以这个实例而言是重复 3 次，这称为非贪婪模式，方法是在正则表达式的查找模式右边增加 ? 符号。

程序实例 ch16_22.py：以非贪婪模式重新设计 ch16_21.py，请读者留意第 12 行的正则表达式的查找模式最右边的 ? 符号。

```
12  pattern = '(son){3,5}?'      # 非贪婪模式
```

执行结果

```
===================== RESTART: D:\Python\ch16\ch16_22.py =====================
查找成功  sonsonson
>>>
```

16-5 正则表达式的特殊字符

为了不在一开始学习正则表达式就太复杂，在前面 4 节只介绍了 \d，同时穿插介绍了一些字符串的查找。我们知道，\d 代表的是数字字符，也就是 0 ～ 9 的阿拉伯数字，如果使用管道 |，\d 相当于下列正则表达式。

```
(0|1|2|3|4|5|6|7|8|9)
```

这一节将针对正则表达式的特殊字符做一个完整的说明。

16-5-1 特殊字符表

字符	使用说明
\d	0 ～ 9 的整数
\D	除了 0 ～ 9 的整数以外的其他字符
\s	空白、定位、Tab 键、换行、换页字符
\S	除了空白、定位、Tab 键、换行、换页字符以外的其他字符
\w	数字、字母和下画线 _ 字符，[A-Za-z0-9_]
\W	除了数字、字母和下画线 _ 字符，[a-Za-Z0-9_] 以外的其他字符

下面是一些使用上述特殊字符的正则表达式的实例说明。

程序实例 ch16_23.py：将一段英文句子的单词分离，同时将英文单词前 4 个字母是“John”的单词分离。笔者设置如下：

```
pattern = '\w+'          # 意义是不限长度的数字、字母和下画线字符当作符合查找
pattern = 'John\w*'      # John 开头后面接 0 到多个数字、字母和下画线字符
```

```
1  # ch16_23.py
2  import re
3  # 测试1将字符串从句子分离
4  msg = 'John, Johnson, Johnnason and Johnnathan will attend my party tonight.'
5  pattern = '\w+'                    # 不限长度的单词
6  txt = re.findall(pattern,msg)      # 返回查找结果
7  print(txt)
8  # 测试2将John开始的字符串分离
9  msg = 'John, Johnson, Johnnason and Johnnathan will attend my party tonight.'
10 pattern = 'John\w*'                # John开头的单词
11 txt = re.findall(pattern,msg)      # 返回查找结果
12 print(txt)
```

执行结果

```
===================== RESTART: D:\Python\ch16\ch16_23.py =====================
['John', 'Johnson', 'Johnnason', 'and', 'Johnnathan', 'will', 'attend', 'my', 'p
arty', 'tonight']
['John', 'Johnson', 'Johnnason', 'Johnnathan']
```

程序实例 ch16_24.py：正则表达式的应用，下列程序重点是第 5 行。

\d+：表示不限长度的数字。

\s：表示空格。

\w+：表示不限长度的数字、字母和下画线字符连续字符。

```
1  # ch16_24.py
2  import re
3
4  msg = '1 cat, 2 dogs, 3 pigs, 4 swans'
5  pattern = '\d+\s\w+'
6  txt = re.findall(pattern,msg)       # 返回查找结果
7  print(txt)
```

执行结果

```
====================== RESTART: D:\Python\ch16\ch16_24.py ======================
['1 cat', '2 dogs', '3 pigs', '4 swans']
```

16-5-2　字符分类

Python 可以使用中括号来设置字符，可参考下列范例。

[a-z]：代表 a ～ z 的小写字符。

[A-Z]：代表 A ～ Z 的大写字符。

[aeiouAEIOU]：代表英文发音的元音字符。

[2-5]：代表 2 ～ 5 的数字。

在字符分类中，中括号内可以不用放上正则表示法的反斜杠 \ 执行，".、?、*、(、) 等字符的转译。例如，[2-5.] 会查找 2 ～ 5 的数字和句点，这个语法不用写成 [2-5\.]。

程序实例 ch16_25.py：查找字符的应用，这个程序首先查找 [aeiouAEIOU]，然后查找 [2-5.]。

```
 1  # ch16_25.py
 2  import re
 3  # 测试1查找[aeiouAEIOU]字符
 4  msg = 'John, Johnson, Johnnason and Johnnathan will attend my party tonight.'
 5  pattern = '[aeiouAEIOU]'
 6  txt = re.findall(pattern,msg)       # 返回查找结果
 7  print(txt)
 8  # 测试2查找[2-5.]字符
 9  msg = '1. cat, 2. dogs, 3. pigs, 4. swans'
10  pattern = '[2-5.]'
11  txt = re.findall(pattern,msg)       # 返回查找结果
12  print(txt)
```

执行结果

```
====================== RESTART: D:\Python\ch16\ch16_25.py ======================
['o', 'o', 'o', 'o', 'a', 'o', 'a', 'o', 'a', 'a', 'i', 'a', 'e', 'a', 'o', 'i']
['.', '2', '.', '3', '.', '4', '.']
```

16-5-3　字符分类的 ^ 字符

在 16-5-2 节字符的处理中，如果在中括号内的左方加上 ^ 字符，意义是查找不在这些字符内的所有字符。

程序实例 ch16_26.py：使用字符分类的 ^ 字符重新设计 ch16_25.py。

```
1  # ch16_26.py
2  import re
3  # 测试1查找不在[aeiouAEIOU]的字符
4  msg = 'John, Johnson, Johnnason and Johnnathan will attend my party tonight.'
5  pattern = '[^aeiouAEIOU]'
6  txt = re.findall(pattern,msg)        # 返回查找结果
7  print(txt)
8  # 测试2查找不在[2-5.]的字符
9  msg = '1. cat, 2. dogs, 3. pigs, 4. swans'
10 pattern = '[^2-5.]'
11 txt = re.findall(pattern,msg)        # 返回查找结果
12 print(txt)
```

执行结果

```
===================== RESTART: D:\Python\ch16\ch16_26.py =====================
['J', 'h', 'n', ',', ' ', 'J', 'h', 'n', 's', 'n', ',', ' ', 'J', 'h', 'n', 'n',
 's', 'n', ' ', 'n', 'd', ' ', 'J', 'h', 'n', 'n', 't', 'h', 'n', ' ', 'w', 'l',
 'l', ' ', 't', 't', 'n', 'd', ' ', 'm', 'y', ' ', 'p', 'r', 't', 'y', ' ', 't',
 'n', 'g', 'h', 't', '.']
['1', ' ', 'c', 'a', 't', ',', ' ', ' ', 'd', 'o', 'g', 's', ',', ' ', ' ', 'p',
 'i', 'g', 's', ',', ' ', ' ', 's', 'w', 'a', 'n', 's']
```

上述第一个测试结果不会出现 [aeiouAEIOU] 字符，第二个测试结果不会出现 [2-5.] 字符。

16-5-4 正则表示法的 ^ 字符

这个 ^ 字符与 16-5-3 节的 ^ 字符完全相同，但是用在不一样的地方，意义不同。在正则表示法中起始位置加上 ^ 字符，表示是正则表示法的字符串必须出现在被查找字符串的起始位置，如果查找成功才算成功。

程序实例 ch16_27.py：正则表示法 ^ 字符的应用，测试 1 字符串 John 是在最前面所以可以得到查找结果，测试 2 字符串 John 不是在最前面，结果查找失败返回空字符串。

```
1  # ch16_27.py
2  import re
3  # 测试1查找John字符串在最前面
4  msg = 'John will attend my party tonight.'
5  pattern = '^John'
6  txt = re.findall(pattern,msg)        # 返回查找结果
7  print(txt)
8  # 测试2查找John字符串不是在最前面
9  msg = 'My best friend is John'
10 pattern = '^John'
11 txt = re.findall(pattern,msg)        # 返回查找结果
12 print(txt)
```

执行结果

```
===================== RESTART: D:\Python\ch16\ch16_27.py =====================
['John']
[]
```

16-5-5 正则表示法的 $ 字符

正则表示法的末端放置 $ 字符时，表示是正则表示法的字符串必须出现在被查找字符串的最后位置，如果查找成功才算成功。

程序实例 ch16_28.py：正则表示法 $ 字符的应用，测试 1 是查找字符串结尾是非英文字符、数字和下画线字符，由于结尾字符是“.”，所以返回所查找到的字符。测试 2 是查找字符串结尾是非英文字符、数字和下画线字符，由于结尾字符是“8”，所以返回查找结果是空字符串。测试 3 是查找字符串结尾是数字字符，由于结尾字符是“8”，所以返回查找结果“8”。测试 4 是查找字符串结尾是数字字符，由于结尾字符是“.”，所以返回查找结果是空字符串。

```
1  # ch16_28.py
2  import re
3  # 测试1查找最后字符是非英文字母数字和下画线字符
4  msg = 'John will attend my party 28 tonight.'
5  pattern = '\W$'
6  txt = re.findall(pattern,msg)       # 返回查找结果
7  print(txt)
8  # 测试2查找最后字符是非英文字母数字和下画线字符
9  msg = 'I am 28'
10 pattern = '\W$'
11 txt = re.findall(pattern,msg)       # 返回查找结果
12 print(txt)
13 # 测试3查找最后字符是数字
14 msg = 'I am 28'
15 pattern = '\d$'
16 txt = re.findall(pattern,msg)       # 返回查找结果
17 print(txt)
18 # 测试4查找最后字符是数字
19 msg = 'I am 28 year old.'
20 pattern = '\d$'
21 txt = re.findall(pattern,msg)       # 返回查找结果
22 print(txt)
```

执行结果

```
==================== RESTART: D:\Python\ch16\ch16_28.py ====================
['.']
[]
['8']
[]
```

也可以将 16-5-4 节的 ^ 字符和 $ 字符混合使用，这时如果既要符合开始字符串也要符合结束字符串，所以被查找的句子一定要只有一个字符串。

程序实例 ch16_29.py：查找开始到结束都是数字的字符串，字符串内容只要有非数字字符就算查找失败。测试 2 中由于中间有非数字字符，所以查找失败。读者应留意程序第 5 行的正则表达式的写法。

```
1  # ch16_29.py
2  import re
3  # 测试1查找开始到结尾皆是数字的字符串
4  msg = '09282028222'
5  pattern = '^\d+$'
6  txt = re.findall(pattern,msg)       # 返回查找结果
7  print(txt)
8  # 测试2查找开始到结尾皆是数字的字符串
9  msg = '0928tuyr990'
10 pattern = '^\d+$'
11 txt = re.findall(pattern,msg)       # 返回查找结果
12 print(txt)
```

执行结果

```
==================== RESTART: D:\Python\ch16\ch16_29.py ====================
['09282028222']
[]
```

16-5-6 单一字符使用通配符“.”

通配符（wildcard）“.”表示可以查找除了换行字符以外的所有字符，但是只限定一个字符。

程序实例 ch16_30.py：通配符的应用。查找一个通配符加上 at，在下列输出中第 4 个结果，由于 at 符合，Python 自动加上空格符。第 6 个结果由于只能加上一个字符，所以查找结果是 lat。

```
1  # ch16_30.py
2  import re
3  msg = 'cat hat sat at matter flat'
4  pattern = '.at'
5  txt = re.findall(pattern,msg)       # 返回查找结果
6  print(txt)
```

执行结果

```
===================== RESTART: D:\Python\ch16\ch16_30.py =====================
['cat', 'hat', 'sat', ' at', 'mat', 'lat']
```

如果查找的是真正的“.”字符，必须使用反斜杠“\.”。

16-5-7 所有字符使用通配符“.*”

若是将 16-3-7 节所介绍的“.”字符与“*”组合，可以查找所有字符，意义是查找 0 到多个通配符（换行字符除外）。

程序实例 ch16_31.py：查找所有字符“.*”的组合应用。

```
1  # ch16_31.py
2  import re
3
4  msg = 'Name: Jiin-Kwei Hung Address: 8F, Nan-Jing E. Rd, Taipei'
5  pattern = 'Name: (.*) Address: (.*)'
6  txt = re.search(pattern,msg)       # 返回查找结果
7  Name, Address = txt.groups()
8  print("Name:     ", Name)
9  print("Address: ", Address)
```

执行结果

```
===================== RESTART: D:\Python\ch16\ch16_31.py =====================
Name:     Jiin-Kwei Hung
Address:  8F, Nan-Jing E. Rd, Taipei
```

16-5-8 换行字符的处理

使用 16-5-7 节的概念用“.*”查找时碰上换行字符，查找就停止。Python 的 re 模块提供参数 re.DOTALL，功能是查找时包括换行字符，可以将此参数放在 search( )、findall( ) 或 compile( )。

程序实例 ch16_32.py：测试 1 是查找换行字符以外的字符，测试 2 是查找含换行字符的所有字符。由于测试 2 包含换行字符，所以输出时，换行字符主导分两行输出。

```
1  # ch16_32.py
2  import re
3  #测试1查找除了换行字符以外的字符
4  msg = 'Name: Jiin-Kwei Hung \nAddress: 8F, Nan-Jing E. Rd, Taipei'
5  pattern = '.*'
6  txt = re.search(pattern,msg)           # 返回查找不含换行字符结果
7  print("测试1输出: ", txt.group())
8  #测试2查找包括换行字符
9  msg = 'Name: Jiin-Kwei Hung \nAddress: 8F, Nan-Jing E. Rd, Taipei'
10 pattern = '.*'
11 txt = re.search(pattern,msg,re.DOTALL) # 返回查找含换行字符结果
12 print("测试2输出: ", txt.group())
```

执行结果

```
===================== RESTART: D:\Python\ch16\ch16_32.py =====================
测试1输出:  Name: Jiin-Kwei Hung
测试2输出:  Name: Jiin-Kwei Hung
Address: 8F, Nan-Jing E. Rd, Taipei
>>>
```

16-6 MatchObject 对象

16-2 节已经讲解使用 re.search( ) 查找字符串，查找成功时可以产生 MatchObject 对象，这里将先介绍另一个查找对象的方法 re.match( )，这个方法查找成功后也将产生 MatchObject 对象。接着本节会分成几小节，再讲解 MatchObject 对象的几个重要的方法（method）。

16-6-1 re.match( )

这本书已经讲解了查找字符串中最重要的两个方法 re.search( ) 和 re.findall( )。re 模块的另一个方法是 re.match( )，这个方法其实和 re.search( ) 相同，差异是 re.match( ) 是只查找比对字符串开始的字，如果失败就算失败。re.search( ) 则是查找整个字符串。至于 re.match( ) 查找成功会返回 MatchObject 对象，若是查找失败会返回 None，这部分与 re.search( ) 相同。

程序实例 ch16_33.py：re.match( ) 的应用。测试 1 是将 John 放在被查找字符串的最前面，测试 2 没有将 John 放在被查找字符串的最前面。

```
1  # ch16_33.py
2  import re
3  #测试1查找使用re.match()
4  msg = 'John will attend my party tonight.'  # John是第一个字符串
5  pattern = 'John'
6  txt = re.match(pattern,msg)                 # 返回查找结果
7  if txt != None:
8      print("测试1输出: ", txt.group())
9  else:
10     print("测试1查找失败")
11 #测试2查找使用re.match()
12 msg = 'My best friend is John.'             # John不是第一个字符串
13 txt = re.match(pattern,msg,re.DOTALL)       # 返回查找结果
14 if txt != None:
15     print("测试2输出: ", txt.group())
16 else:
17     print("测试2查找失败")
```

执行结果

```
==================== RESTART: D:\Python\ch16\ch16_33.py ====================
测试1输出:  John
测试2查找失败
>>>
```

16-6-2 MatchObject 几个重要的方法

当使用 re.search( ) 或 re.match( ) 查找成功时，会产生 MatchObject 对象。

程序实例 ch16_34.py：看看 MatchObject 对象是什么。

```
1  # ch16_34.py
2  import re
3  #测试1查找使用re.match()
4  msg = 'John will attend my party tonight.'
5  pattern = 'John'
6  txt = re.match(pattern,msg)                # re.match()
7  if txt != None:
8      print("使用re.match()输出MatchObject对象:  ", txt)
9  else:
10     print("测试1查找失败")
11 #测试1查找使用re.search()
12 txt = re.search(pattern,msg)               # re.search()
13 if txt != None:
14     print("使用re.search()输出MatchObject对象: ", txt)
15 else:
16     print("测试1查找失败")
```

执行结果

```
==================== RESTART: D:\Python\ch16\ch16_34.py ====================
使用re.match()输出MatchObject对象:   <_sre.SRE_Match object; span=(0, 4), match='John'>
使用re.search()输出MatchObject对象:  <_sre.SRE_Match object; span=(0, 4), match='John'>
>>>
```

从上述可知，当使用 re.match( ) 和 re.search( ) 都查找成功时，二者的 MatchObject 对象内容是相同的。span 是注明成功查找字符串的起始位置和结束位置，从此处可以知道起始索引位置是 0，结束索引位置是 4。match 则是注明成功查找的字符串内容。

Python 提供下列取得 MatchObject 对象内容的重要方法。

方法	说明
group( )	可返回查找到的字符串，本章已有许多实例说明
end( )	可返回查找到字符串的结束位置
start( )	可返回查找到字符串的起始位置
span( )	可返回查找到字符串的（起始，结束）位置

程序实例 ch16_35.py：分别使用 re.match( ) 和 re.search( ) 查找字符串 John，获得成功查找字符串时，分别用 start( )、end( ) 和 span( ) 方法列出字符串出现的位置。

```
1  # ch16_35.py
2  import re
3  #测试1查找使用re.match()
4  msg = 'John will attend my party tonight.'
5  pattern = 'John'
6  txt = re.match(pattern,msg)              # re.match()
7  if txt != None:
8      print("查找成功字符串的起始索引位置 :  ", txt.start())
9      print("查找成功字符串的结束索引位置 :  ", txt.end())
10     print("查找成功字符串的结束索引位置 :  ", txt.span())
11 #测试2查找使用re.search()
12 msg = 'My best friend is John.'
13 txt = re.search(pattern,msg)             # re.search()
14 if txt != None:
15     print("查找成功字符串的起始索引位置 :  ", txt.start())
16     print("查找成功字符串的结束索引位置 :  ", txt.end())
17     print("查找成功字符串的结束索引位置 :  ", txt.span())
```

执行结果

```
===================== RESTART: D:\Python\ch16\ch16_35.py =====================
查找成功字符串的起始索引位置 :   0
查找成功字符串的结束索引位置 :   4
查找成功字符串的结束索引位置 :   (0, 4)
查找成功字符串的起始索引位置 :   18
查找成功字符串的结束索引位置 :   22
查找成功字符串的结束索引位置 :   (18, 22)
>>>
```

16-7　抢救 CIA 情报员——sub( ) 方法

Python re 模块内的 sub( ) 方法可以用新的字符串取代原本字符串的内容。

16-7-1　一般的应用

sub( ) 方法的基本使用语法如下：

```
result = re.sub(pattern, newstr, msg)     # msg 是整个要处理的字符串或句子
```

pattern 是要查找的字符串，如果查找成功则用 newstr 取代，同时成功取代的结果返回给 result 变量，如果查找到多个相同字符串，这些字符串将全部被取代，需留意原先 msg 内容将不会改变。如果查找失败则将 msg 内容返回给 result 变量，当然 msg 内容也不会改变。

程序实例 ch16_36.py：这是字符串取代的应用，测试 1 是发现两个字符串被成功取代（Eli Nan 被 Kevin Thomson 取代），同时列出取代结果。测试 2 是取代失败，所以 txt 与原 msg 内容相同。

```
1  # ch16_36.py
2  import re
3  #测试1取代使用re.sub()结果成功
4  msg = 'Eli Nan will attend my party tonight. My best friend is Eli Nan'
5  pattern = 'Eli Nan'                      # 要查找字符串
6  newstr = 'Kevin Thomson'                 # 新字符串
7  txt = re.sub(pattern,newstr,msg)         # 如果找到则取代
8  if txt != msg:                           # 如果txt与msg内容不同表示取代成功
9      print("取代成功: ", txt)               # 列出成功取代结果
10 else:
11     print("取代失败: ", txt)               # 列出失败取代结果
12 #测试2取代使用re.sub()结果失败
13 pattern = 'Eli Thomson'                  # 要查找字符串
14 txt = re.sub(pattern,newstr,msg)         # 如果找到则取代
15 if txt != msg:                           # 如果txt与msg内容不同表示取代成功
16     print("取代成功: ", txt)               # 列出成功取代结果
17 else:
18     print("取代失败: ", txt)               # 列出失败取代结果
```

执行结果

```
===================== RESTART: D:\Python\ch16\ch16_36.py =====================
取代成功:  Kevin Thomson will attend my party tonight. My best friend is Kevin Thomson
取代失敗:  Eli Nan will attend my party tonight. My best friend is Eli Nan
```

16-7-2 抢救 CIA 情报员

社会上有太多需要保护当事人隐私权利的场合，例如，情报机构在内部文件中不可直接将情报员的名字列出来，历史上太多这类实例造成情报员的牺牲，这时可以使用 *** 代替**姓名**。使用 Python 的正则表示法，可以轻松协助我们完成这方面的工作。这一节将先给出程序代码，然后解析此程序。

程序实例 ch16_37.py：将 CIA 情报员的名字，用名字的第一个字母和 *** 取代。

```
1  # ch16_37.py
2  import re
3  # 使用隐藏文字执行取代
4  msg = 'CIA Mark told CIA Linda that secret USB had given to CIA Peter.'
5  pattern = r'CIA (\w)\w*'          # 要查找CIA + 空一格后的名字
6  newstr = r'\1***'                 # 新字符串使用隐藏文字
7  txt = re.sub(pattern,newstr,msg)  # 执行取代
8  print("取代成功: ", txt)           # 列出取代结果
```

执行结果

```
====================== RESTART: D:\Python\ch16\ch16_37.py ======================
取代成功:  M*** told L*** that secret USB had given to P***.
```

上述程序第一个关键是第 5 行，这一行将查找 CIA 字符串外加空一格后出现不限长度的字符串（可以是英文大小写或数字或下画线所组成）。括号内的（\w）代表必须只有一个字符，同时小括号代表这是一个分组（group），由于整行只有一个括号所以知道这是第一个分组，同时只有一个分组，括号外的 \w* 表示可以有 0 到多个字符。所以（\w）\w* 相当于是 1 到多个字符组成的单词，同时存在分组 1。

上述程序第 6 行的 \1 代表用分组 1 找到的第 1 个字母当作字符串开头，后面 *** 则是接在第 1 个字母后的字符。对 CIA Mark 而言，所找到的第一个字母是 M，所以取代的结果是 M***。对 CIA Linda 而言，所找到的第一个字母是 L，所以取代的结果是 L***。对 CIA Peter 而言，所找到的第一个字母是 P，所以取代的结果是 P***。

16-8 处理比较复杂的正则表示法

有一个正则表示法内容如下：

```
pattern = r((\d{2}|\(\d{2}\))?(\s|-)?\d{8}(\s*(ext|x|ext.)\s*\d{3,5})?)
```

其实相信大部分读者看到上述正则表示法，就想放弃了，坦白地说它的确是复杂的，不过不用担心，下面将一步一步解析让它变简单。

16-8-1　将正则表达式拆成多行字符串

在 3-4-2 节有介绍可以使用 3 个单引号（或是双引号）将过长的字符串拆成多行表达，这个概念也可以应用于正则表达式，当我们适当拆解后，可以为每一行加上注释，整个正则表达式就变得简单了。若是将上述 pattern 拆解成下列表示法，就变得简单了。

```
pattern = r'''(
    (\d{2}|\(\d{2}\))?              # 区域号码
    (\s|-)?                         # 区域号码与电话号码的分隔符号
    \d{8}                           # 电话号码
    (\s*(ext|ext.)\s*\d{2,4})?      # 2~4位数的分机号码
    )'''
```

接下来笔者分别解释相信读者就可以了解了。第 1 行区域号码是两位数，可以接收有括号的区域号码，也可以接收没有括号的区域号码，例如，02 或（02）都可以。第 2 行是设置区域号码与电话号码间的字符，可以接收空格符或 – 字符当作分隔符。第 3 行是设置 8 位数数字的电话号码。第 4 行是分机号码，分机号码可以用 ext 或 ext. 当作起始字符，空一定格数，然后接收 2 ～ 4 位数的分机号码。

16-8-2　re.VERBOSE

使用 Python 时，如果想在正则表达式中加上注释，可参考 16-8-1 节，必须配合使用 re.VERBOSE 参数，然后将此参数放在 search( )、findall( ) 或 compile( ) 中。

程序实例 ch16_38.py：查找市区电话号码的应用，这个程序可以查找下列格式的电话号码。

```
12345678                    # 没有区域号码
02 12345678                 # 区域号码与电话号码间没有空格
02-12345678                 # 区域号码与电话号码间使用 - 分隔
(02)-12345678               # 区域号码有小括号
02-12345678 ext 123         # 有分机号
02-12345678 ext. 123        # 有分机号,ext.右边有 .
```

```
1   # ch16_38.py
2   import re
3
4   msg = '''02-88223349, (02)-26669999, 02-29998888 ext 123,
5            12345678, 02 33887766 ext. 12222'''
6   pattern = r'''(
7       (\d{2}|\(\d{2}\))?                          # 区域号码
8       (\s|-)?                                     # 区域号码与电话号码的分隔符
9       \d{8}                                       # 电话号码
10      (\s*(ext|ext.)\s*\d{2,4})?                  # 2~4位数的分机号码
11      )'''
12  phoneNum = re.findall(pattern, msg, re.VERBOSE)     # 返回查找结果
13  print(phoneNum)
```

执行结果

```
==================== RESTART: D:\Python\ch16\ch16_38.py ====================
[('02-88223349', '02', '-', '', ''), ('(02)-26669999', '(02)', '-', '', ''), ('0
2-29998888 ext 123', '02', '-', ' ext 123', 'ext'), ('12345678', '', '', '', '
'), ('02 33887766 ext. 1222', '02', ' ', ' ext. 1222', 'ext.')]
```

16-8-3　电子邮件地址的查找

在字处理过程中，必须在文件内将电子邮件地址解析出来也很常见，下面是这方面的应用。下列是 pattern 内容。

```
pattern = r'''(
    [a-zA-Z0-9_.]+                  # 使用者账号
    @                               # @符号
    [a-zA-Z0-9-.]+                  # 主机域名domain
    [\.]                            # .符号
    [a-zA-Z]{2,4}                   # 可能是com或edu或其他
    ([\.])?                         # .符号，也可能无，特别是美国
    ([a-zA-Z]{2,4})?                # 国别
    )'''
```

第 1 行用户账号常用的有 a ～ z 字符、A ～ Z 字符、0 ～ 9 数字、下画线 _、点 .。第 2 行是 @ 符号。第 3 行是主机域名，常用的有 a ～ z 字符、A ～ Z 字符、0 ～ 9 数字、分隔符 -、点 .。第 4 行是点 . 符号。第 5 行最常见的是 com 或 edu，也可能是 cc 或其他，通常由 2 ～ 4 个字符组成，常用的有 a ～ z 字符、A ～ Z 字符。第 6 行是点 . 符号，在美国通常只要前 5 行就够了，但是在其他国家则常常需要此字段，所以此字段后面是 ? 字符。第 7 行通常是国别，例如，中国是 cn、日本是 ja，常用的有 a ～ z 字符、A ～ Z 字符。

程序实例 ch16_39.py：电子邮件地址的查找。

```
1   # ch16_39.py
2   import re
3
4   msg = '''txt@deepstone.com.tw kkk@gmail.com'''
5   pattern = r'''(
6       [a-zA-Z0-9_.]+                  # 使用者账号
7       @                               # @符号
8       [a-zA-Z0-9-.]+                  # 主机域名domain
9       [\.]                            # .符号
10      [a-zA-Z]{2,4}                   # 可能是com或edu或其他
11      ([\.])?                         # .符号，也可能无，特别是美国
12      ([a-zA-Z]{2,4})?                # 国别
13      )'''
14  eMail = re.findall(pattern, msg, re.VERBOSE)    # 返回查找结果
15  print(eMail)
```

执行结果

```
===================== RESTART: D:\Python\ch16\ch16_39.py =====================
[('txt@deepstone.com.tw', '', ''), ('kkk@gmail.com', '', '')]
```

16-8-4　re.IGNORECASE/re.DOTALL/re.VERBOSE

在 16-3-9 节介绍了 re.IGNORECASE 参数，在 16-5-8 节介绍了 re.DOTALL 参数，在 16-8-2 节介绍了 re.VERBOSE 参数，可以分别在 re.search( )、re.findall( )、re.match( ) 或是 re.compile( ) 方法内使用它们，可是一次只能放置一个参数，如果想要一次放置多个参数，应如何处理？方法是使用 16-3-4 节的管道 |，例如，可以使用下列方式。

```
datastr = re.search(pattern, msg, re.IGNORECASE|re.DOTALL|re.VERBOSE)
```

其实这一章已经讲解了相当多的正则表达式的知识了，未来读者在写论文、做研究或职场上相

信会有帮助。如果读者仍觉不足，可以自行到 Python 官网获得更多正则表达式的知识。

习题

1. 中国大陆手机号码格式是 xxx-xxxx-xxxx，x 代表数字，请重新设计 ch16_1.py，判断号码是否为中国大陆手机号码。除了原先有两组测试数据外，需另增加一组号码 133-1234-1234 做测试。（16-1 节）

```
====================== RESTART: D:\Python\ex\ex16_1.py ======================
I love Ming-Chi: 是中国大陆手机号码 False
0932-999-199:    是中国大陆手机号码 False
133-1234-1234:   是中国大陆手机号码 True
```

2. 请建立下列文件 ex16_2.txt，内容如下。

```
我喜欢看小龙女与杨过，不仅因为小龙女美丽，杨过在戏中
所扮演的角色更是让我喜欢。
```

请读者参考 ch16_2.py 设计查找字符串**小龙女**，**杨过**，同时列出这个字符串出现的次数。这个程序应该采取交互式设计，程序执行时要求输入要查找的字符串，然后列出查找结果，接着询问是否继续查找，是（y 或 Y）则继续，输入其他字符就是否，则程序结束。

其实如果使用上述语句分析一部小说各个人物出现的次数，就可以知道哪些人物是主角，哪些人物是配角。（16-1 节）

```
====================== RESTART: D:\Python\ex\ex16_2.py ======================
请输入与查找字符串 : 杨过
所查找字符串 杨过 共出现 2 次

是否继续,输入Y或y则程序继续
= y
请输入与查找字符串 : 小龙女
所查找字符串 小龙女 共出现 2 次

是否继续,输入Y或y则程序继续
= y
请输入与查找字符串 : 洪锦魁
所查找字符串 洪锦魁 共出现 0 次

是否继续,输入Y或y则程序继续
= n
```

3. 请重新设计 ch16_20.py，使用下列 pattern 做测试。（16-4 节）

（1）'（son）{2,}'

（2）'（son）{,5}'

```
====================== RESTART: D:\Python\ex\ex16_3.py ======================
以下用(son){2,}做测试
查找失败  None
查找成功  sonson
查找成功  sonsonson
查找成功  sonsonsonson
查找成功  sonsonsonsonson
以下用(son){,5}做测试
查找成功  son
查找成功  sonson
查找成功  sonsonson
查找成功  sonsonsonson
查找成功  sonsonsonsonson
```

4. 请进入本书 ch14 目录，将扩展名是 txt 的文件打印出来，将 ch14_10 .py ～ ch14_19.py 等 10 个文件的文件名打印出来。(16-5 节)

```
====================== RESTART: D:\Python\ex\ex16_4.py ======================
打印*.txt
['ansi14_44.txt', 'ch14_15.txt', 'ch14_20.txt', 'ch14_51.txt', 'data36.txt', 'de
st.txt', 'out14_26.txt', 'out14_27.txt', 'out14_28.txt', 'out14_29.txt', 'out14_
30.txt', 'out14_31.txt', 'out35.txt', 'source.txt', 'sse.txt', 'utf14_45.txt', '
utf14_49.txt', 'zenofPython.txt']
打印ch14_10.py - ch14_19.py
['ch14_10.py', 'ch14_11.py', 'ch14_12.py', 'ch14_13.py', 'ch14_14.py', 'ch14_15.
py', 'ch14_16.py', 'ch14_17.py', 'ch14_18.py', 'ch14_19.py']
```

5. 台湾地区有些地方的电话号码是区域号码 2 位数，电话号码是 7 位数，请修改 ch16_38.py，可以接收 7 位数或 8 位数的电话号码，下列是测试数据。(16-8 节)

```
msg = '''02-88223349, (02)-26669999, 02-29998888 ext 123,
      12345678, 02 33887766 ext. 12222,
      02-1234567, 02-2345X789, 23-123456'''
```

你的结果只需列出通过测试的电话号码。

```
====================== RESTART: D:/Python/ex/ex16_5.py ======================
02-88223349
(02)-26669999
02-29998888 ext 123
 12345678
02 33887766 ext. 1222
02-1234567
```

6. 重新设计 ch16_39.py，请在第 4 行内加上你的电子邮件地址，另外再加上其他两个邮件地址，及一个不符合规定的邮件地址，请将输出结果由列表内的元组元素分离出来，处理成下列方式，下列是测试地址字符串。(16-8 节)

```
msg = '''txt@deepstone.com.tw kkk@gmail.com
abc@me.com mymail@qq.com abc@abc@abc'''
```

下列是执行结果。

```
====================== RESTART: D:/Python/ex/ex16_6.py ======================
txt@deepstone.com.tw
kkk@gmail.com
abc@me.com
mymail@qq.com
```

17

第 1 7 章

用 Python 处理图像文件

本章摘要

在 2020 年，高画质的手机将成为个人标准配备，也许你可以使用许多影像软件处理手机所拍摄的相片，本章将介绍如何用 Python 处理这些照片。本章将使用 Pillow 模块，所以请先导入此模块。

```
pip install pillow
```

注意，在程序设计中需导入的是 PIL 模块，主要原因是要向旧版 Python Image Library 兼容，如下所示。

```
from PIL import ImageColor
```

17-1 认识 Pillow 模块的 RGBA

在 Pillow 模块中 RGBA 分别代表红色（Red）、绿色（Green）、蓝色（Blue）和透明度（Alpha），这 4 个与颜色有关的数值组成**元组**（tuple），每个数值为 0 ～ 255。如果 Alpha 的值是 255，代表完全不透明，值越小透明度越高。其实它的色彩使用方式与 HTML 相同，其他有关颜色的细节可参考附录 D。

17-1-1 getrgb( )

这个函数可以将颜色符号或字符串转为元组，在这里可以使用英文名称，例如“red”；色彩数值，例如 #00ff00：rgb 函数，例如 rgb（0, 255,0）；或以百分比代表颜色的 rgb 函数，例如 rgb（0%,100%,0%）。这个函数在使用时，如果字符串无法被解析判别，将造成 ValueError 异常。这个函数的使用格式如下。

```
(r, g, b) = getrgb(color)              # 返回色彩元组
```

程序实例 ch17_1.py：使用 getrgb( ) 方法返回色彩的元组。

```
1  # ch17_1.py
2  from PIL import ImageColor
3
4  print(ImageColor.getrgb("#0000ff"))
5  print(ImageColor.getrgb("rgb(0, 0, 255)"))
6  print(ImageColor.getrgb("rgb(0%, 0%, 100%)"))
7  print(ImageColor.getrgb("Blue"))
8  print(ImageColor.getrgb("blue"))
```

执行结果

```
==================== RESTART: D:/Python/ch17/ch17_1.py ====================
(0, 0, 255)
(0, 0, 255)
(0, 0, 255)
(0, 0, 255)
(0, 0, 255)
>>>
```

17-1-2 getcolor( )

getcolor( ) 的功能基本上与 getrgb( ) 相同，它的使用格式如下。

```
(r, g, b) = getcolor(color, "mode")          # 返回色彩元组
(r, g, b, a) = getcolor(color, "mode")       # 返回色彩元组
```

mode 若是填写 "RGBA" 则可返回 RGBA 元组，如果填写 "RGB" 则返回 RGB 元组。

程序实例 ch17_2.py：测试使用 getcolor( ) 函数，了解返回值。

```
1  # ch17_2.py
2  from PIL import ImageColor
3
4  print(ImageColor.getcolor("#0000ff", "RGB"))
5  print(ImageColor.getcolor("rgb(0, 0, 255)", "RGB"))
6  print(ImageColor.getcolor("Blue", "RGB"))
7  print(ImageColor.getcolor("#0000ff", "RGBA"))
8  print(ImageColor.getcolor("rgb(0, 0, 255)", "RGBA"))
9  print(ImageColor.getcolor("Blue", "RGBA"))
```

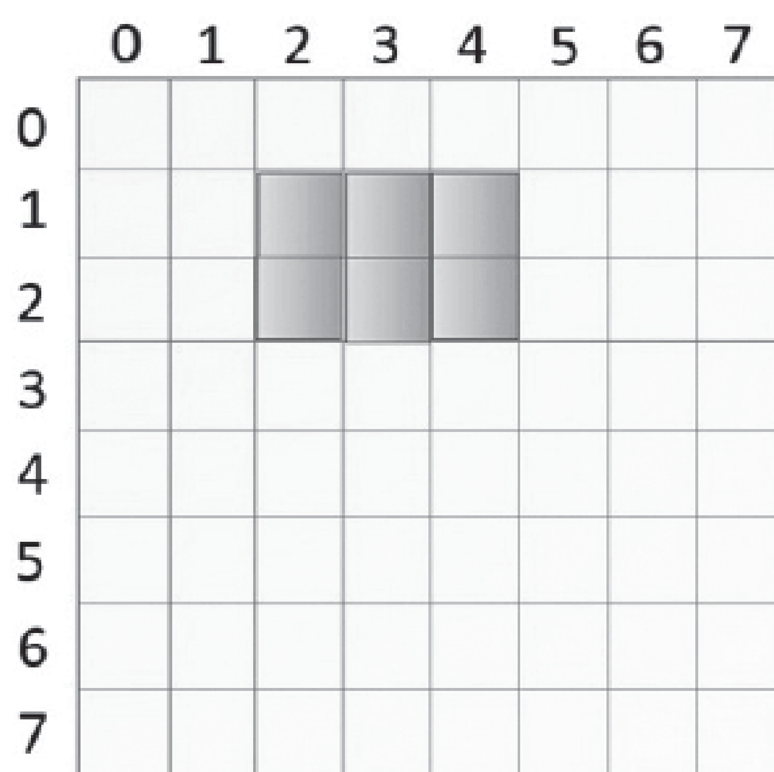
执行结果

```
==================== RESTART: D:\Python\ch17\ch17_2.py ====================
(0, 0, 255)
(0, 0, 255)
(0, 0, 255)
(0, 0, 255, 255)
(0, 0, 255, 255)
(0, 0, 255, 255)
>>>
```

17-2 Pillow 模块的盒子元组

17-2-1　基本概念

下图是 Pillow 模块的图像坐标的概念。



最左上角的像素（x,y）是（0,0），x 轴像素值往右递增，y 轴像素值往下递增。盒子元组的参数是（left, top, right, bottom），意义如下。

left：盒子左上角的 x 轴坐标。

top：盒子左上角的 y 轴坐标。

right：盒子右下角的 x 轴坐标。

bottom：盒子右下角的 y 轴坐标。

若上图是一张图片，则可以用（2, 1, 4, 2）表示它的盒子元组（box tuple），也就是它的图像坐标。

17-2-2　计算机眼中的图像

上述图像坐标格子的列数和行数称为分辨率（resolution）。例如，我们说某个图像的分辨率是1280×720，表示宽度的格子数有 1280 个，高度的格子数有 720 个。

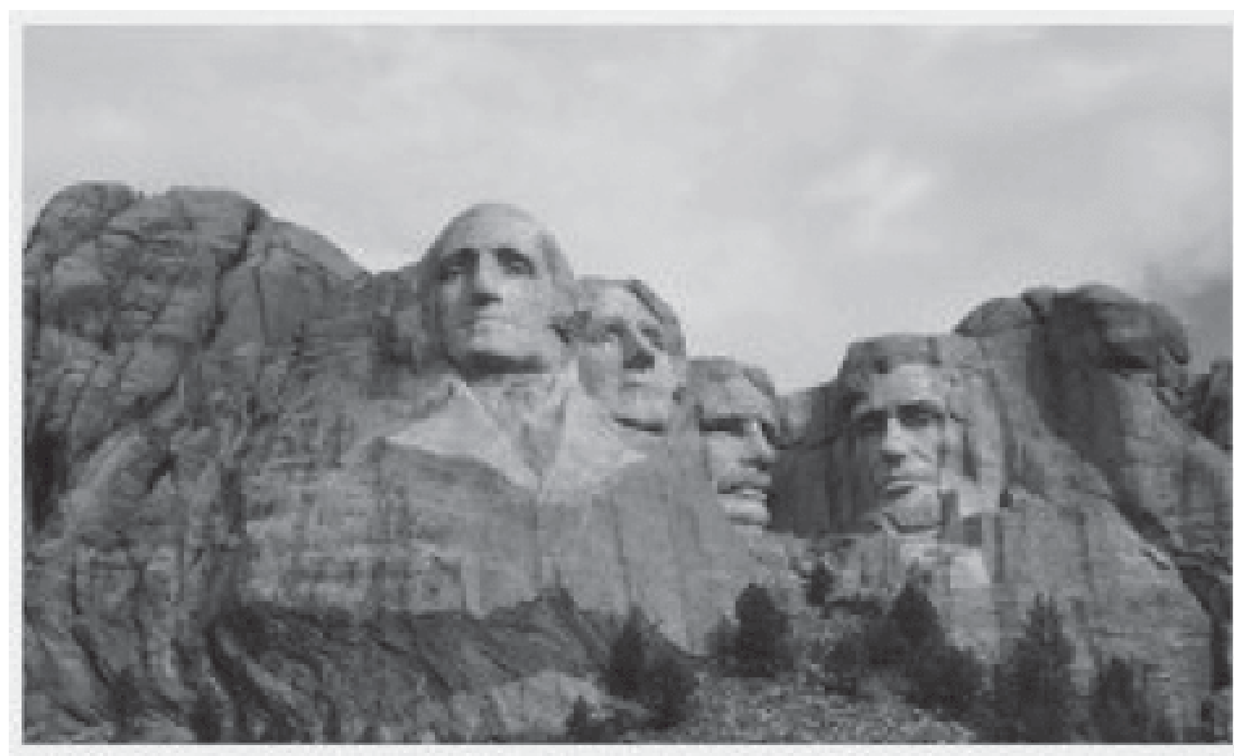
图像坐标的每一个像素可以用颜色值代表，如果是灰阶色彩，可以用 0 ～ 255 的数字表示，0 是最暗的黑色，255 代表白色。也就是说我们可以用一个矩阵（matirix）代表一个灰阶的图。

如果是彩色的图，每个像素是用（R,G,B）代表，R 是 Red、G 是 Green、B 是 Blue，每个颜色也是 0 ～ 255，我们所看到的色彩其实就是由这 3 个原色所组成。如果矩阵每个位置可以存放 3 个元素的元组，我们可以用含 3 个颜色值（R, G, B）的元组代表这个像素，这时可以只用一个数组（matrix）代表此彩色图像。如果我们坚持一个数组只放一个颜色值，可以用 3 个矩阵（matrix）代表此彩色图像。

在人工智能的图像识别中，很重要的是找出图像特征，所使用的卷积运算就是使用这些图像的矩阵数字，执行更进一步的运算。

17-3　图像的基本操作

本节使用的图像文件是 rushmore.jpg，在 ch17 文件夹中可以找到，此图片内容如下。



17-3-1　打开图像对象

可以使用 open( ) 方法打开一个图像对象，参数是要打开的图像文件名。

17-3-2　图像大小属性

可以使用 size 属性获得图像大小，这个属性可返回图像的宽（width）和高（height）。

程序实例 ch17_3.py：在 ch17 文件夹中有 rushmore.jpg 文件，这个程序会列出此图像文件的宽和高。

```
1  # ch17_3.py
2  from PIL import Image
3
4  rushMore = Image.open("rushmore.jpg")          # 建立Pillow对象
5  print("列出对象类型 : ", type(rushMore))
6  width, height = rushMore.size                  # 获得图像宽度和高度
7  print("宽度 = ", width)
8  print("高度 = ", height)
```

执行结果

```
===================== RESTART: D:\Python\ch17\ch17_3.py =====================
列出对象类型 :  <class 'PIL.JpegImagePlugin.JpegImageFile'>
宽度 =  270
高度 =  161
```

17-3-3　取得图像对象文件名

可以使用 filename 属性获得图像的源文件名称。

程序实例 ch17_4.py：获得图像对象的文件名。

```
1  # ch17_4.py
2  from PIL import Image
3
4  rushMore = Image.open("rushmore.jpg")        # 建立Pillow对象
5  print("列出对象文件名 : ", rushMore.filename)
```

执行结果

```
===================== RESTART: D:\Python\ch17\ch17_4.py =====================
列出对象文件名 :  rushmore.jpg
```

17-3-4　取得图像对象的文件格式

可以使用 format 属性获得图像文件格式（可想成图像文件的扩展名），此外，可以使用 format_description 属性获得更详细的文件格式描述。

程序实例 ch17_5.py：获得图像对象的扩展名与描述。

```
1  # ch17_5.py
2  from PIL import Image
3
4  rushMore = Image.open("rushmore.jpg")        # 建立Pillow对象
5  print("列出对象扩展名 : ", rushMore.format)
6  print("列出对象描述   : ", rushMore.format_description)
```

执行结果

```
===================== RESTART: D:\Python\ch17\ch17_5.py =====================
列出对象扩展名 :  JPEG
列出对象描述   :  JPEG (ISO 10918)
```

17-3-5　存储文件

可以使用 save( ) 方法存储文件，也可以将 jpg 文件转存成 png 或 gif 文件，反之亦可，这些都是图像文件但是以不同格式存储。

程序实例 ch17_6.py：将 rushmore.jpg 转存成 out17_6.png。

```
1  # ch17_6.py
2  from PIL import Image
3
4  rushMore = Image.open("rushmore.jpg")        # 建立Pillow对象
5  rushMore.save("out17_6.png")
```

执行结果 在 ch17 文件夹将可以看到所建的 out17_6.png。

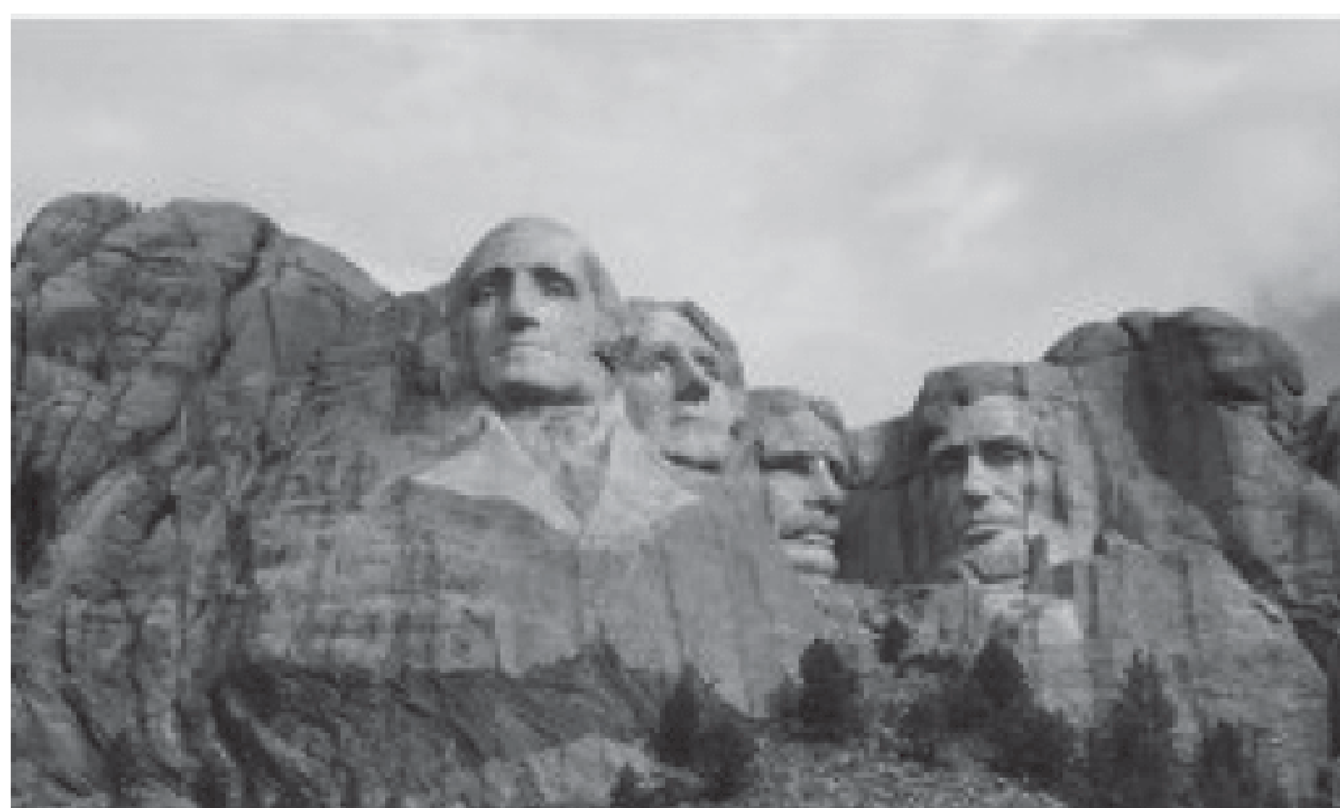
17-3-6 屏幕显示图像

可以使用 show( ) 方法直接显示图像，在 Windows 操作系统下可以使用此方法调用 Windows 照片查看器显示图像。

程序实例 ch17_6_1.py：在屏幕上显示 rushmore.jpg 图像。

```
1  # ch17_6_1.py
2  from PIL import Image
3
4  rushMore = Image.open("rushmore.jpg")        # 建立Pillow对象
5  rushMore.show()
```

执行结果



17-3-7 建立新的图像对象

可以使用 new( ) 方法建立新的图像对象，它的语法格式如下。

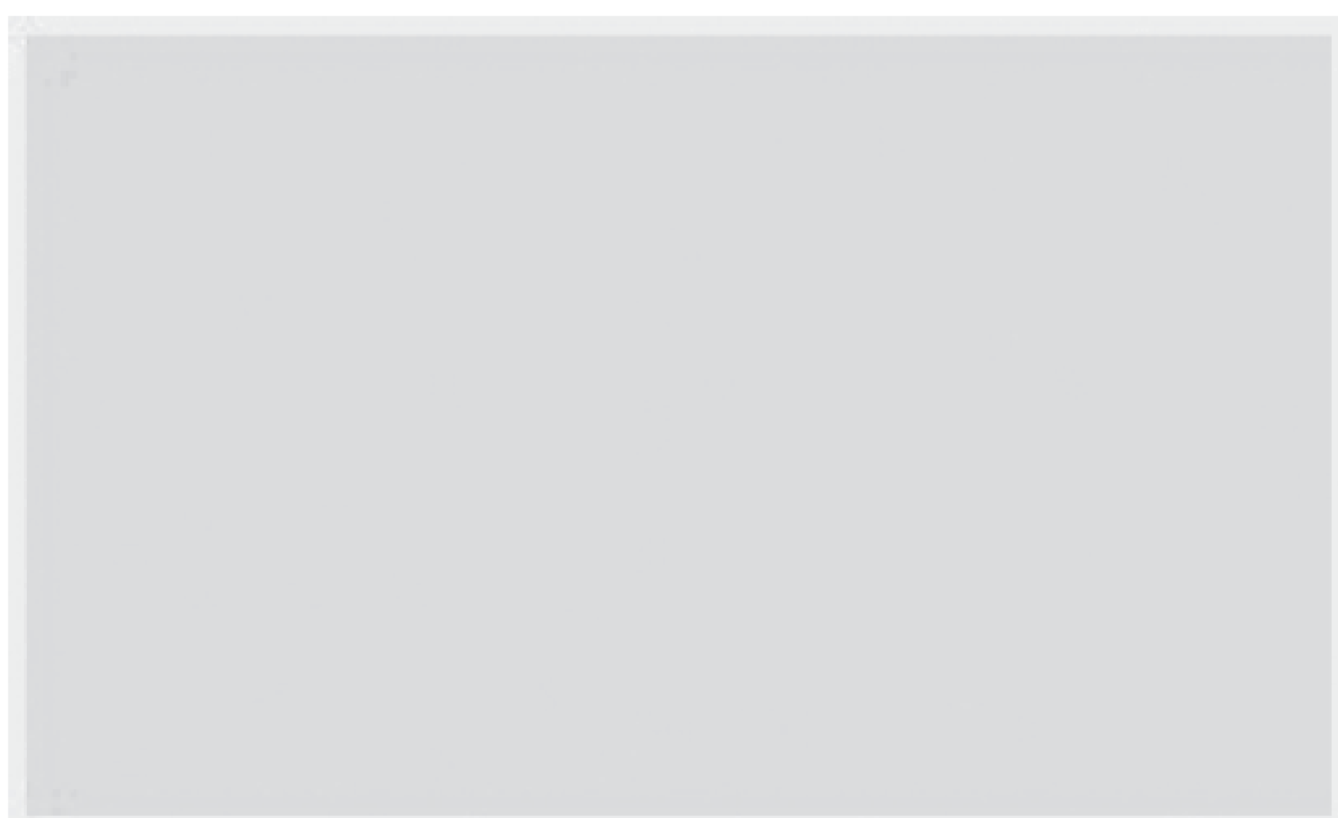
```
new(mode, size, color=0)
```

mode 可以有多种设置，一般建议用 "RGBA"（建立 png 文件）或 "RGB"（建立 jpg 文件）即可。size 参数是一个元组（tuple），可以设置新图像的宽度和高度。color 默认是黑色，不过可以参考附录 D 建立不同的颜色。

程序实例 ch17_7.py：建立一个水蓝色（aqua）的图像文件 out17_7.jpg。

```
1  # ch17_7.py
2  from PIL import Image
3
4  pictObj = Image.new("RGB", (300, 180), "aqua")  # 建立aqua颜色图像
5  pictObj.save("out17_7.jpg")
```

执行结果　在 ch17 文件夹可以看到下列 out17_7.jpg 文件。



程序实例 ch17_8.py：建立一个透明的黑色的图像文件 out17_8.png。

```
1  # ch17_8.py
2  from PIL import Image
3
4  pictObj = Image.new("RGBA", (300, 180))      # 建立完全透明图像
5  pictObj.save("out17_8.png")
```

执行结果　文件打开后因为透明，看不出任何效果。

17-4　图像的编辑

17-4-1　更改图像大小

Pillow 模块提供 resize( ) 方法可以调整图像大小，它的使用语法如下。

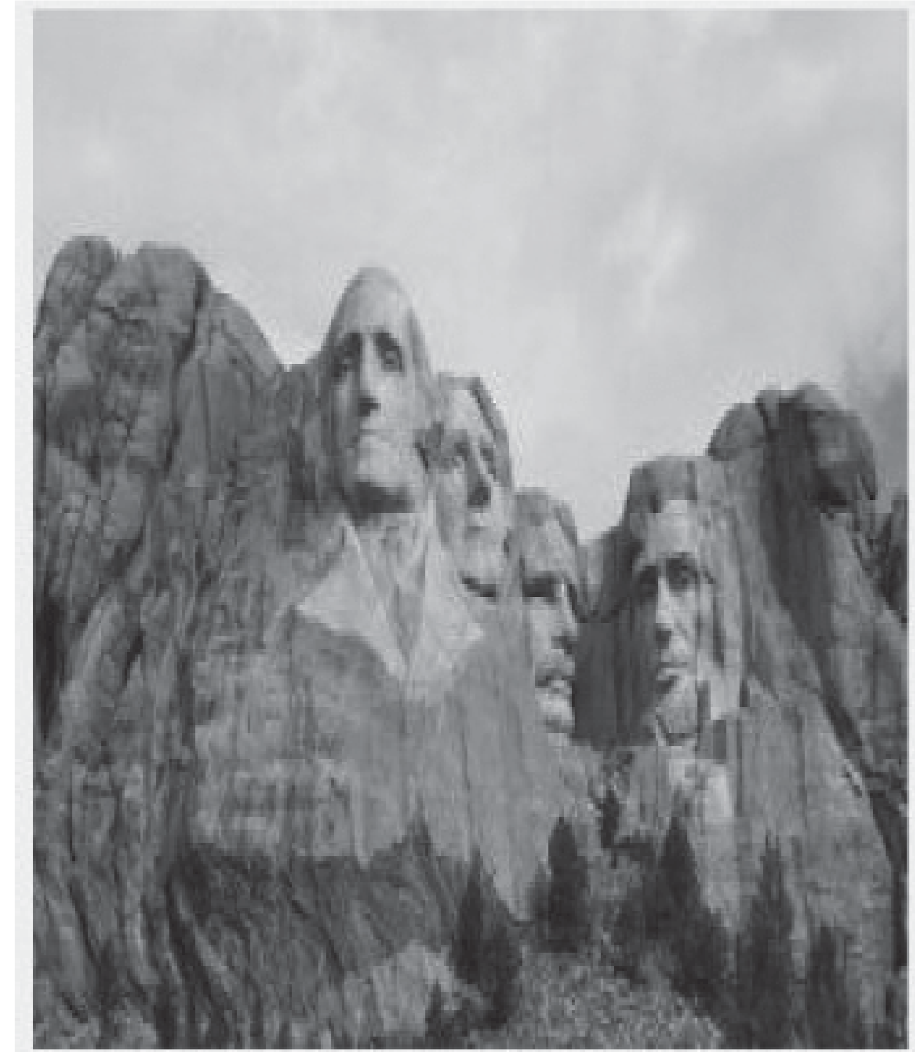
resize ((width, heigh), Image.BILINEAR))　# 双线取样法，也可以省略

第一个参数是新图像的宽与高，以元组表示，是整数。第二个参数主要是设置更改图像所使用的方法，常见的除上述方法外，也可以设置 Image.NEAREST（最低质量）、Image.ANTIALIAS（最高质量）、Image.BISCUBIC（三次方取样法），一般可以省略。

程序实例 ch17_9.py：分别将图片的宽度与高度增加为原先的 2 倍。

```
1  # ch17_9.py
2  from PIL import Image
3
4  pict = Image.open("rushmore.jpg")              # 建立Pillow对象
5  width, height = pict.size
6  newPict1 = pict.resize((width*2, height))     # 宽度是2倍
7  newPict1.save("out17_9_1.jpg")
8  newPict2 = pict.resize((width, height*2))     # 高度是2倍
9  newPict2.save("out17_9_2.jpg")
```

执行结果　下列分别是 out17_9_1.jpg（左）与 out17_9_2.jpg（右）的执行结果。

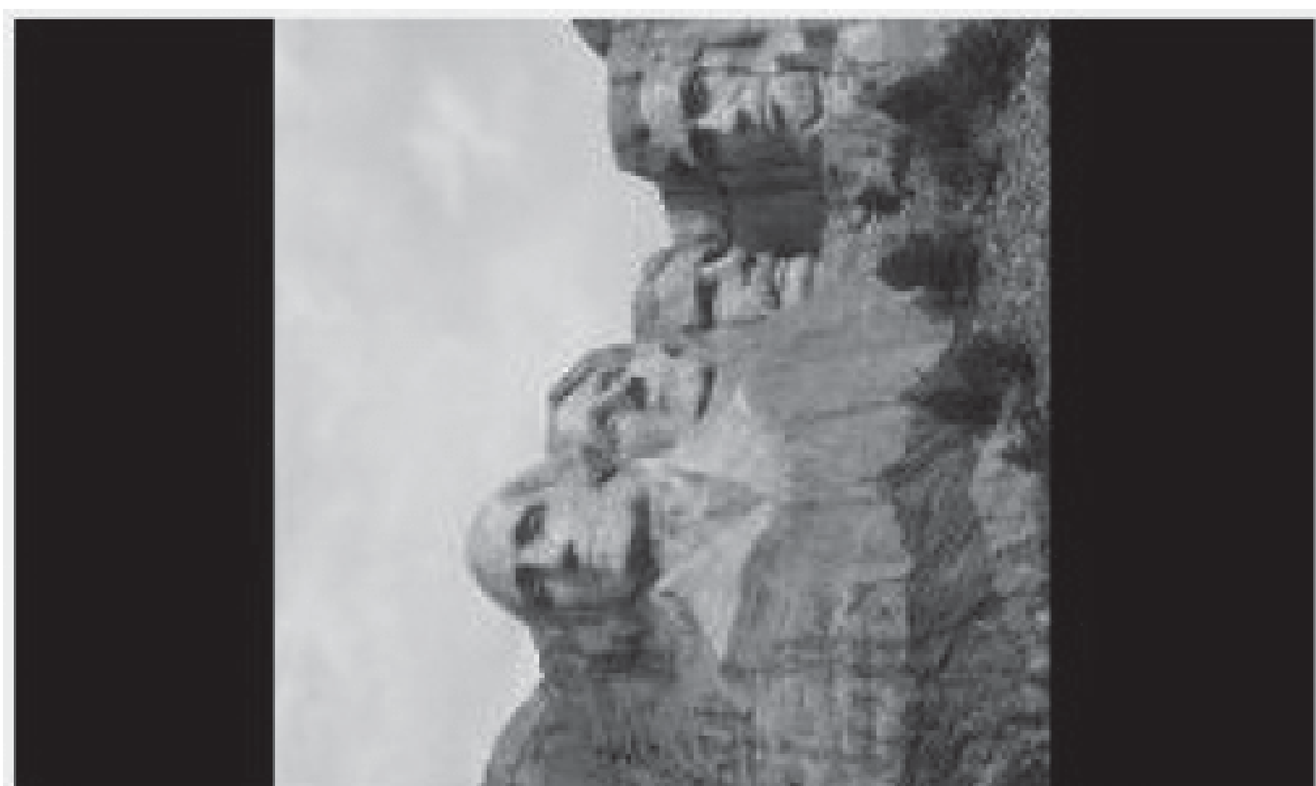
17-4-2　图像的旋转

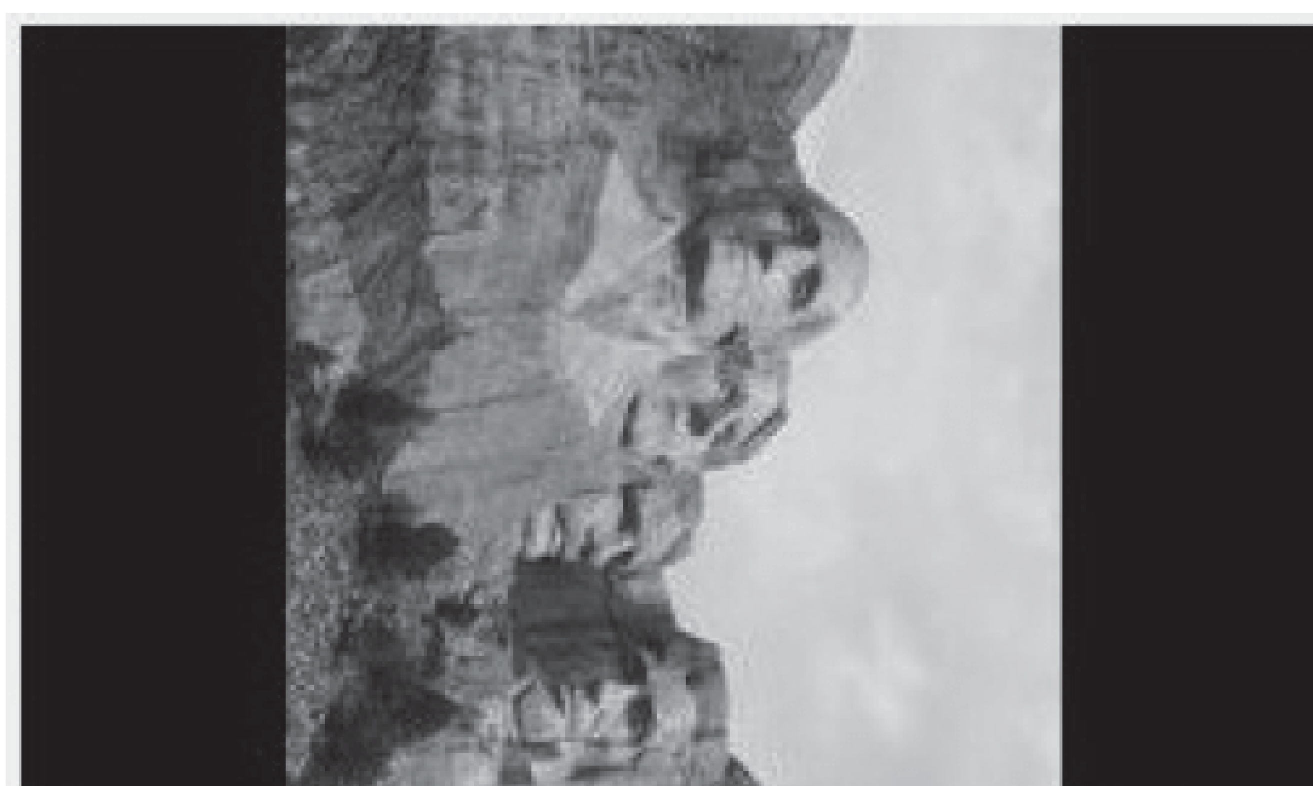
Pillow 模块提供 rotate( ) 方法可以逆时针旋转图像，如果旋转 90° 或 270° ，图像的宽度与高度会有变化，但图像本身比例不变，多的部分以黑色图像替代，如果是其他角度则图像维持不变。

程序实例 ch17_10.py：将图像分别旋转 90° 、180° 和 270° 。

```
1  # ch17_10.py
2  from PIL import Image
3
4  pict = Image.open("rushmore.jpg")              # 建立Pillow对象
5  pict.rotate(90).save("out17_10_1.jpg")         # 旋转90°
6  pict.rotate(180).save("out17_10_2.jpg")        # 旋转180°
7  pict.rotate(270).save("out17_10_3.jpg")        # 旋转270°
```

执行结果 下列分别是旋转 90° 、180° 、270° 的结果。





在使用 rotate( ) 方法时也可以增加第 2 个参数 expand=True，如果有这个参数会放大图像，让

整个图像显示，多余部分用黑色填满。

程序实例 ch17_11.py：没有使用 expand=True 参数与使用此参数的比较。

```
1  # ch17_11.py
2  from PIL import Image
3
4  pict = Image.open("rushmore.jpg")                      # 建立Pillow对象
5  pict.rotate(45).save("out17_11_1.jpg")                 # 旋转45°
6  pict.rotate(45, expand=True).save("out17_11_2.jpg")    # 旋转45° 图像扩充
```

执行结果 下列分别是 out17_11_1.jpg 与 out17_11_2.jpg 的图像内容。





17-4-3　图像的翻转

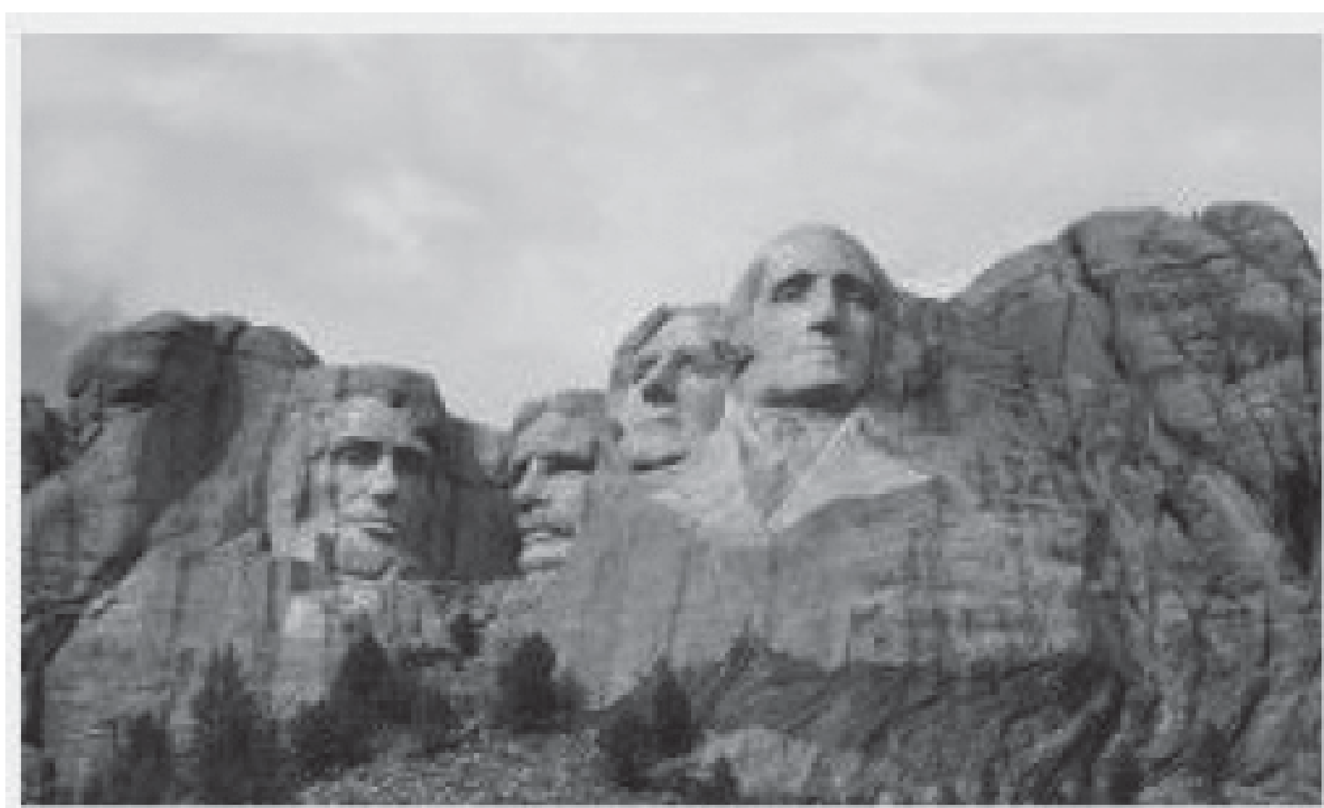
可以使用 transpose( ) 让图像翻转，这个方法使用语法如下。

```
transpose(Image.FLIP_LEFT_RIGHT)     # 图像左右翻转
transpose(Image.FLIP_TOP_BOTTOM)     # 图像上下翻转
```

程序实例 ch17_12.py：图像左右翻转与上下翻转的实例。

```
1  # ch17_12.py
2  from PIL import Image
3
4  pict = Image.open("rushmore.jpg")                                  # 建立Pillow对象
5  pict.transpose(Image.FLIP_LEFT_RIGHT).save("out17_12_1.jpg")       # 左右
6  pict.transpose(Image.FLIP_TOP_BOTTOM).save("out17_12_2.jpg")       # 上下
```

执行结果 下列分别是左右翻转与上下翻转的结果。





17-4-4 图像像素的编辑

Pillow 模块的 getpixel( ) 方法可以取得图像某一位置像素（pixel）的色彩。

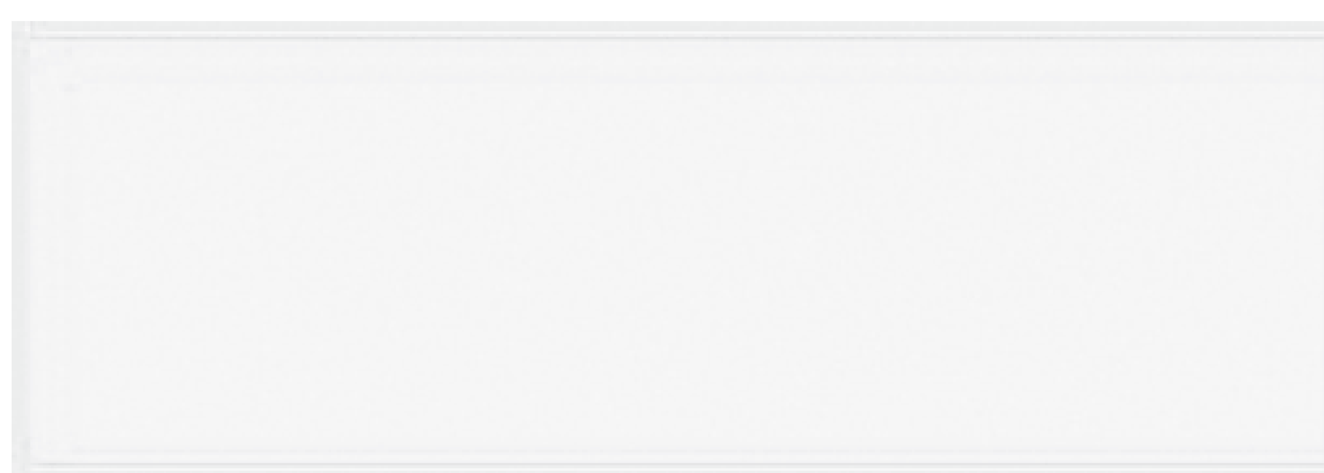
getpixel((x,y))　　　# 参数是元组 (x,y)，这是像素位置

程序实例 ch17_13.py：先建立一个图像，大小是（300,100），色彩是 Yellow，然后列出图像中心点的色彩。最后将图像存储至 out17_13.png。

```
1  # ch17_13.py
2  from PIL import Image
3
4  newImage = Image.new('RGBA', (300, 100), "Yellow")
5  print(newImage.getpixel((150, 50)))        # 打印中心点的色彩
6  newImage.save("out17_13.png")
```

执行结果 下列是执行结果与 out17_13.png 内容。

```
==================== RESTART: D:\Python\ch17\ch17_13.py ====================
(255, 255, 0, 255)
>>>
```



Pillow 模块的 putpixel( ) 方法可以在图像的某一个位置填入色彩，常用的语法如下。

putpixel((x,y), (r, g, b, a))　　　# 两个参数分别是位置与色彩元组

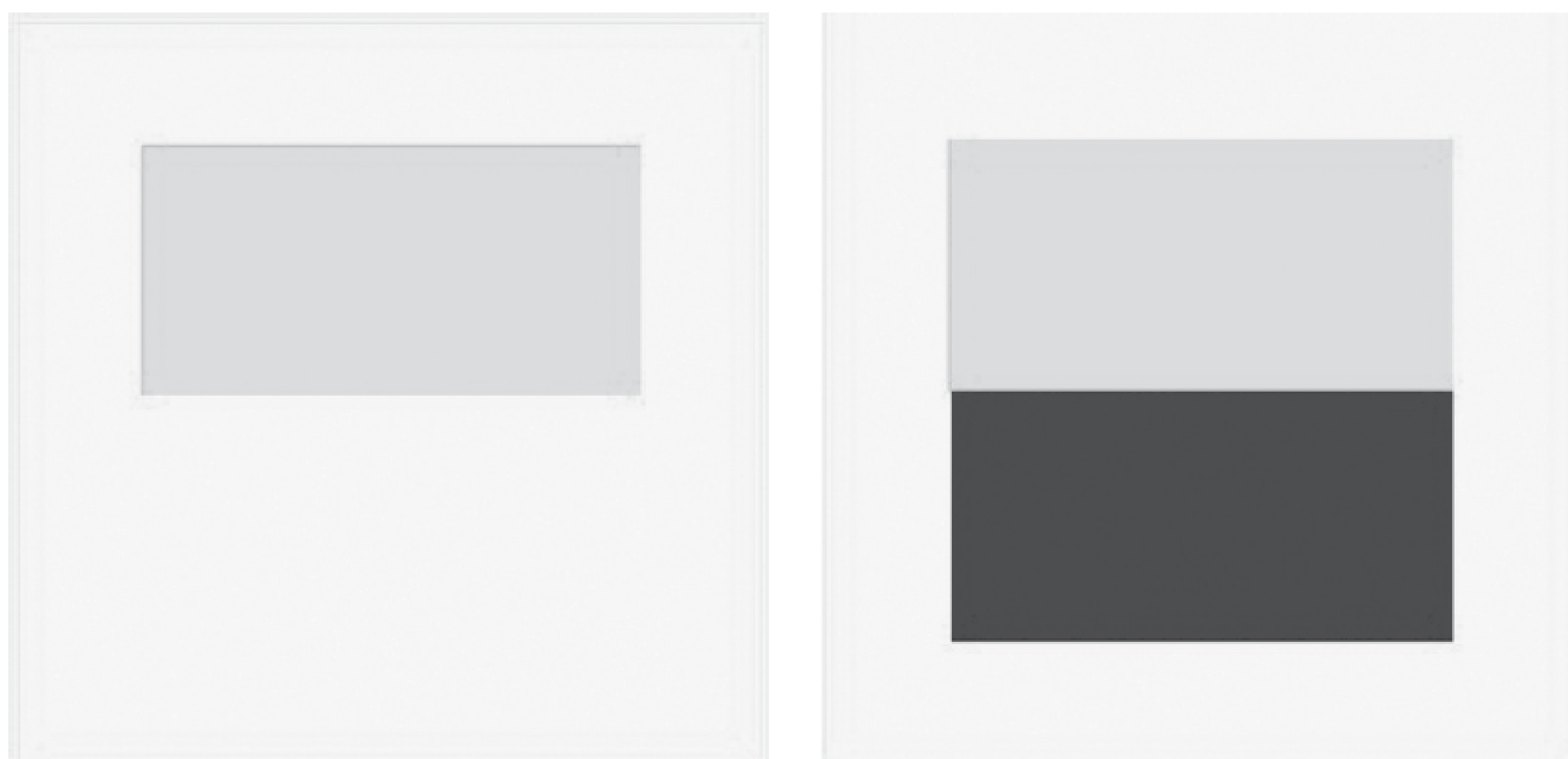
上述色彩元组的值是 0 ～ 255，若省略 a 代表不透明。另外也可以用 17-1-2 节的 getcolor( ) 当作第 2 个参数，用这种方法可以直接用附录 D 的色彩名称填入指定像素位置，例如，下列是填入蓝色（Blue）的方法。

putpixel((x,y), ImageColor.getcolor("Blue", "RGBA"))　# 需先导入 ImageColor

程序实例 ch17_14.py：建立一个 300×300 的图像，底色是黄色（Yellow），然后 (50, 50, 250, 150) 是填入青色（Cyan），此时将上述执行结果存入 out17_14_1.png。然后将蓝色（Blue）填入 (50, 151, 250, 250)，最后将结果存入 out17_14_2.png。

```
1  # ch17_14.py
2  from PIL import Image
3  from PIL import ImageColor
4
5  newImage = Image.new('RGBA', (300, 300), "Yellow")
6  for x in range(50, 251):                                    # x轴区间在50~250
7      for y in range(50, 151):                                # y轴区间在50~150
8          newImage.putpixel((x, y), (0, 255, 255, 255))       # 填青色
9  newImage.save("out17_14_1.png")                             # 第一阶段存档
10 for x in range(50, 251):                                    # x轴区间在50~250
11     for y in range(151, 251):                               # y轴区间在151~250
12         newImage.putpixel((x, y), ImageColor.getcolor("Blue", "RGBA"))
13 newImage.save("out17_14_2.png")                             # 第一阶段存档
```

执行结果 下列分别是第一阶段与第二阶段的执行结果。



17-5 裁切、复制与图像合成

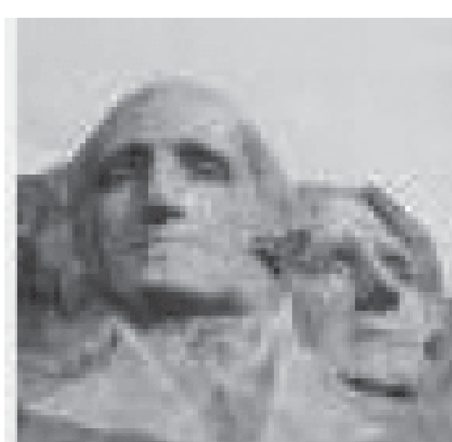
17-5-1 裁切图像

Pillow 模块有提供 crop( ) 方法可以裁切图像，其中参数是一个元组，元组内容是（左 , 上 , 右 , 下）的区间坐标。

程序实例 ch17_15.py：裁切（80, 30, 150, 100）区间。

```
1  # ch17_15.py
2  from PIL import Image
3
4  pict = Image.open("rushmore.jpg")           # 建立Pillow对象
5  cropPict = pict.crop((80, 30, 150, 100))    # 裁切区间
6  cropPict.save("out17_15.jpg")
```

执行结果　下列是 out17_15.jpg 的裁切结果。



17-5-2 复制图像

假设想要执行图像合成处理，为了不破坏原图像内容，建议先保存图像，再执行合成动作。Pillow 模块有提供 copy( ) 方法可以复制图像。

程序实例 ch17_16.py：复制图像，再将所复制的图像存储。

```
1  # ch17_16.py
2  from PIL import Image
3
4  pict = Image.open("rushmore.jpg")           # 建立Pillow对象
5  copyPict = pict.copy()                      # 复制
6  copyPict.save("out17_16.jpg")
```

执行结果 下列是 out17_16.jpg 的执行结果。



17-5-3 图像合成

Pillow 模块有提供 paste( ) 方法可以合成图像，它的语法如下：

底图图像 .paste(插入图像 , (x,y))　　　　　　# (x,y) 元组是插入位置

程序实例 ch17_17.py：使用 rushmore.jpg 图像，为这个图像复制一份 copyPict，裁切一份 cropPict，将 cropPict 合成至 copyPict 内两次，将结果存入 out17_17.jpg。

```
1  # ch17_17.py
2  from PIL import Image
3
4  pict = Image.open("rushmore.jpg")                  # 建立Pillow对象
5  copyPict = pict.copy()                             # 复制
6  cropPict = copyPict.crop((80, 30, 150, 100))       # 裁切区间
7  copyPict.paste(cropPict, (20, 20))                 # 第一次合成
8  copyPict.paste(cropPict, (20, 100))                # 第二次合成
9  copyPict.save("out17_17.jpg")                      # 存储
```

执行结果



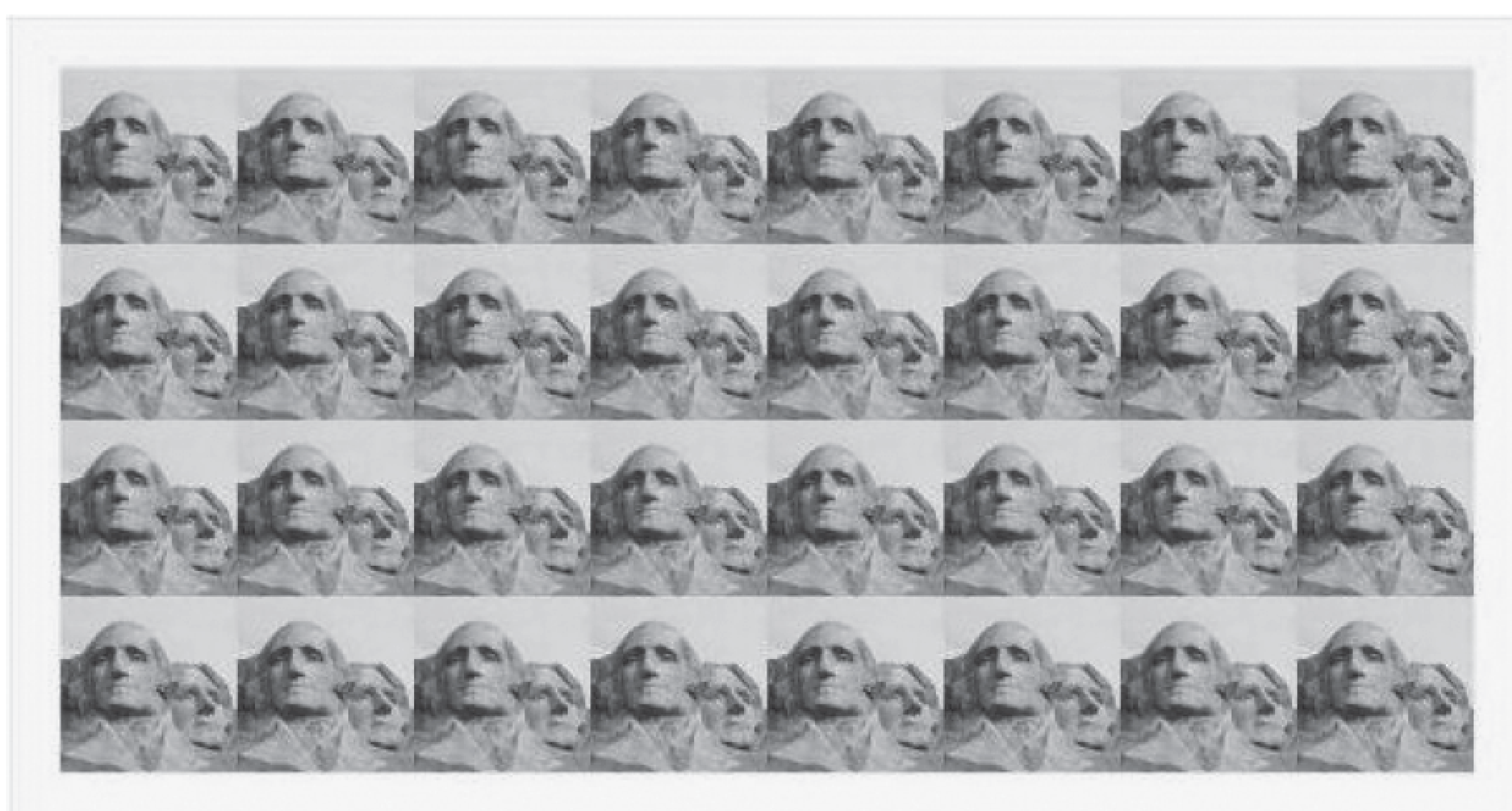
17-5-4 将裁切图片填满图像区间

在 Windows 操作系统使用中常看到图片填满某一区间，其实可以用双层循环完成这个工作。

程序实例 ch17_18.py：将一个裁切的图片填满某一个图像区间，最后存储此图像，在这个图像设计中，笔者也设置了留白区间，这个区间是图像建立时的颜色。

```
1  # ch17_18.py
2  from PIL import Image
3
4  pict = Image.open("rushmore.jpg")                    # 建立Pillow对象
5  copyPict = pict.copy()                               # 复制
6  cropPict = copyPict.crop((80, 30, 150, 100))         # 裁切区间
7  cropWidth, cropHeight = cropPict.size                # 获得裁切区间的宽与高
8
9  width, height = 600, 320                             # 新图像宽与高
10 newImage = Image.new('RGB', (width, height), "Yellow")  # 建立新图像
11 for x in range(20, width-20, cropWidth):              # 双层循环合成
12     for y in range(20, height-20, cropHeight):
13         newImage.paste(cropPict, (x, y))             # 合成
14
15 newImage.save("out17_18.jpg")                        # 存储
```

执行结果



17-6　图像滤镜

Pillow 模块内有 ImageFilter 模块，使用此模块可以增加 filter( ) 方法为图片加上滤镜效果。此方法的参数意义如下。

BLUR：模糊。

CONTOUR：轮廓。

DETAIL：细节增强。

EDGE_ENHANCE：边缘增强。

EDGE_ENHANCE_MORE：深度边缘增强。

EMBOSS：浮雕效果。

FIND_EDGES：边缘信息。

SMOOTH：平滑效果。

SMOOTH_MORE：深度平滑效果。

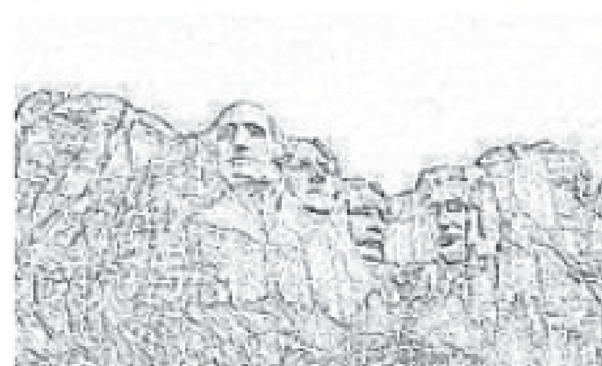
SHARPEN：锐利化效果。

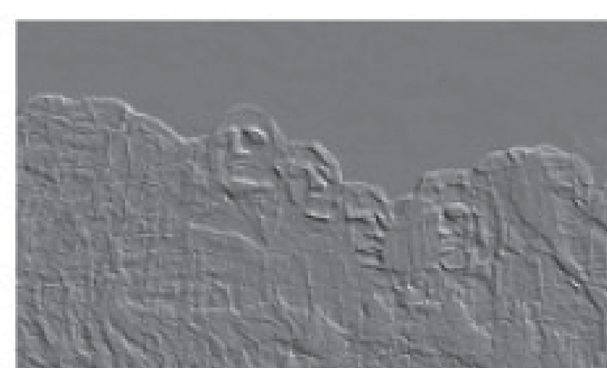
程序实例 ch17_19.py：使用滤镜处理图片。

```
1  # ch17_19.py
2  from PIL import Image
3  from PIL import ImageFilter
4  rushMore = Image.open("rushmore.jpg")        # 建立Pillow对象
5  filterPict = rushMore.filter(ImageFilter.BLUR)
6  filterPict.save("out17_19_BLUR.jpg")
7  filterPict = rushMore.filter(ImageFilter.CONTOUR)
8  filterPict.save("out17_19_CONTOUR.jpg")
9  filterPict = rushMore.filter(ImageFilter.EMBOSS)
10 filterPict.save("out17_19_EMBOSS.jpg")
11 filterPict = rushMore.filter(ImageFilter.FIND_EDGES)
12 filterPict.save("out17_19_FIND_EDGES.jpg")
```

执行结果


BLUR


CONTOUR


EMBOSS


FIND_EDGES

17-7 在图像内绘制图案

Pillow 模块内有一个 ImageDraw 模块，可以利用此模块绘制点（Points）、线（Lines）、矩形（Rectangles）、椭圆（Ellipses）、多边形（Polygons）。

在图像内建立图案对象方式如下。

```
from PIL import Image, ImageDraw
newImage = Image.new('RGBA', (300, 300), "Yellow")  # 建立300×300黄色底的图像
drawObj = ImageDraw.Draw(newImage)
```

17-7-1 绘制点

ImageDraw 模块的 point( ) 方法可以绘制点，语法如下。

```
point([(x1,y1), … (xn,yn)], fill)          # fill 是设置颜色
```

第一个参数是由元组（tuple）组成的列表，(x,y）是要绘制的点坐标。fill 可以是 RGBA( ) 或是直接指定颜色。

17-7-2 绘制线条

ImageDraw 模块的 line( ) 方法可以绘制线条，语法如下。

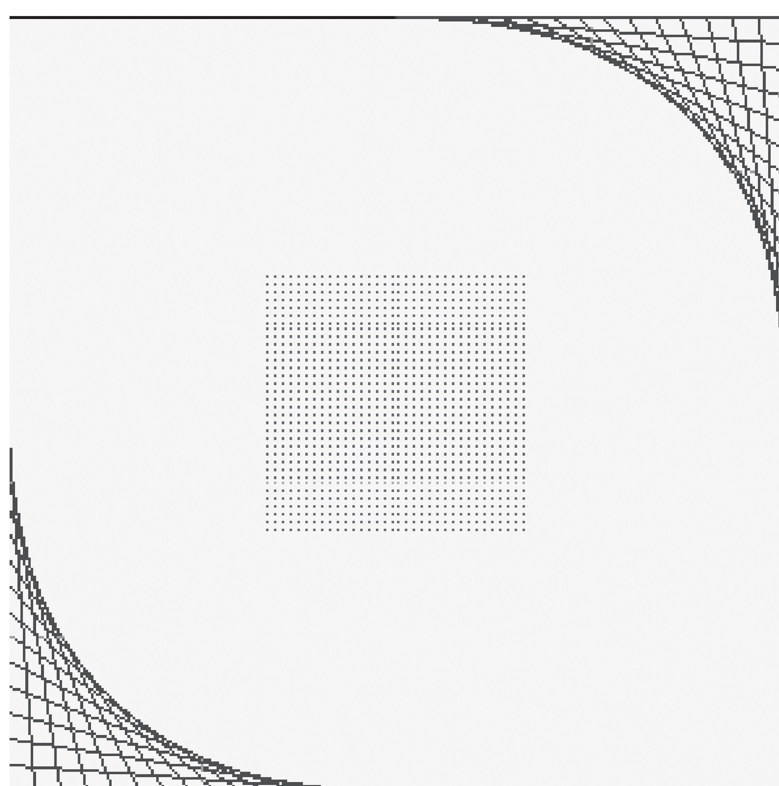
line([(x1,y1), … (xn,yn)], **width**, fill)　# **width** 是宽度，默认是 1

第一个参数是由元组（tuple）组成的列表，(x,y) 是要绘制线条的点坐标，如果多于两个点，则这些点会串接起来。fill 可以是 RGBA( ) 或是直接指定颜色。

程序实例 ch17_20.py：绘制点和线条的应用。

```
1  # ch17_20.py
2  from PIL import Image, ImageDraw
3
4  newImage = Image.new('RGBA', (300, 300), "Yellow")  # 建立300×300黄色底的图像
5  drawObj = ImageDraw.Draw(newImage)
6
7  # 绘制点
8  for x in range(100, 200, 3):
9      for y in range(100, 200, 3):
10         drawObj.point([(x,y)], fill='Green')
11
12 # 绘制线条，绘外框线
13 drawObj.line([(0,0), (299,0), (299,299), (0,299), (0,0)], fill="Black")
14 # 绘制右上角美工线
15 for x in range(150, 300, 10):
16     drawObj.line([(x,0), (300,x-150)], fill="Blue")
17 # 绘制左下角美工线
18 for y in range(150, 300, 10):
19     drawObj.line([(0,y), (y-150,300)], fill="Blue")
20 newImage.save("out17_20.png")
```

执行结果



17-7-3　绘制圆或椭圆

ImageDraw 模块的 ellipse( ) 方法可以绘制圆或椭圆，语法如下。

ellipse((left,top,right,bottom), fill, **outline**)　# **outline** 是外框颜色

第一个参数是由元组（tuple）组成的，(left,top,right,bottom) 是包住圆或椭圆的矩形左上角与右下角的坐标。fill 可以是 RGBA( ) 或是直接指定颜色，outline 可选择是否加上。

17-7-4　绘制矩形

ImageDraw 模块的 rectangle( ) 方法可以绘制矩形，语法如下。

rectangle((left,top,right,bottom), fill, **outline**) # **outline** 是外框颜色

第一个参数是由元组（tuple）组成的，(left,top,right,bottom) 是矩形左上角与右下角的坐标。

fill 可以是 RGBA( ) 或是直接指定颜色，outline 可选择是否加上。

17-7-5 绘制多边形

ImageDraw 模块的 polygon( ) 方法可以绘制多边形，语法如下。

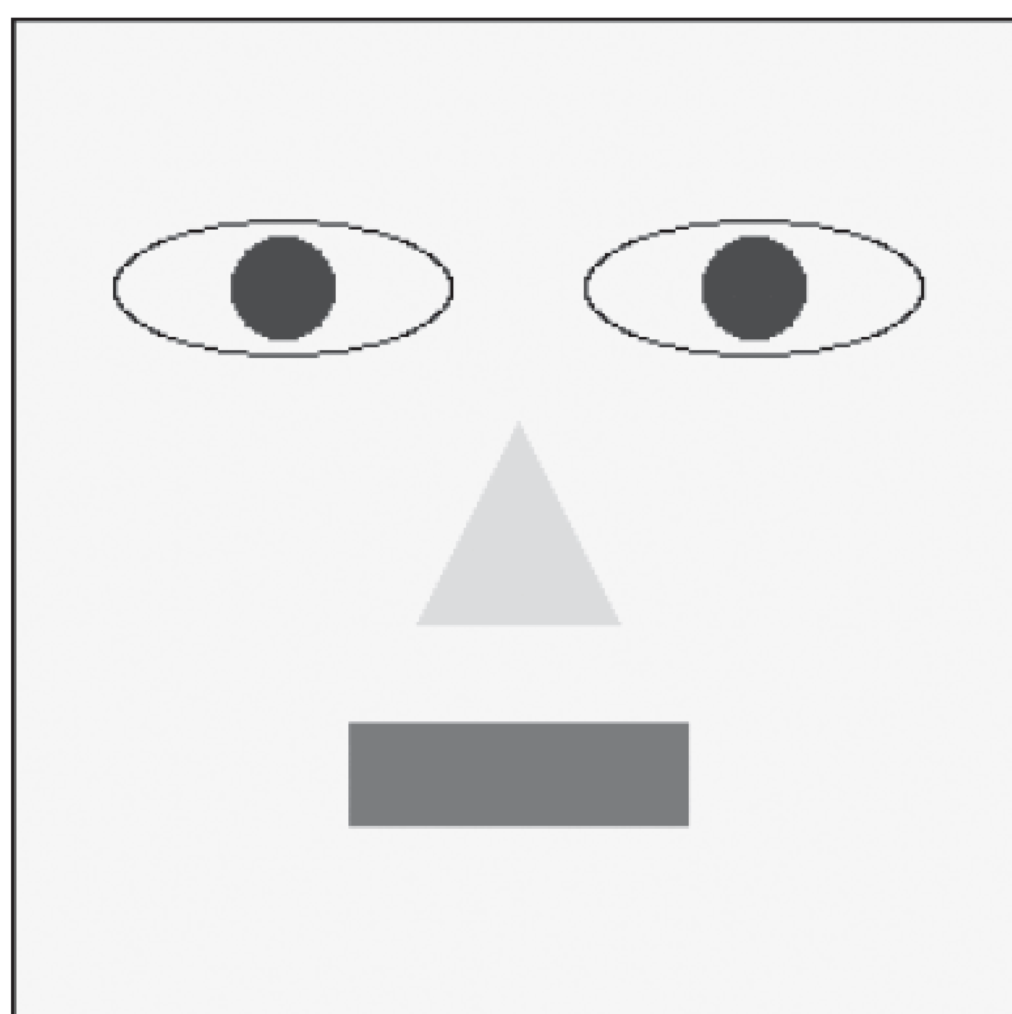
polygon([(x1,y1), … (xn,yn)], fill, outline)　　# outline 是外框颜色

第一个参数是由元组（tuple）组成的列表，（x,y）是要绘制多边形的点坐标，在此填上多边形各端点坐标。fill 可以是 RGBA( ) 或是直接指定颜色，outline 可选择是否加上。

程序实例 ch17_21.py：设计一个图案。

```
1  # ch17_21.py
2  from PIL import Image, ImageDraw
3
4  newImage = Image.new('RGBA', (300, 300), 'Yellow')  # 建立300×300黄色底的图像
5  drawObj = ImageDraw.Draw(newImage)
6
7  drawObj.rectangle((0,0,299,299), outline='Black')    # 图像外框线
8  drawObj.ellipse((30,60,130,100),outline='Black')     # 左眼外框
9  drawObj.ellipse((65,65,95,95),fill='Blue')           # 左眼
10 drawObj.ellipse((170,60,270,100),outline='Black')    # 右眼外框
11 drawObj.ellipse((205,65,235,95),fill='Blue')         # 右眼
12 drawObj.polygon([(150,120),(180,180),(120,180),(150,120)],fill='Aqua') # 鼻子
13 drawObj.rectangle((100,210,200,240), fill='Red')     # 嘴
14 newImage.save("out17_21.png")
```

执行结果



17-8 在图像内填写文字

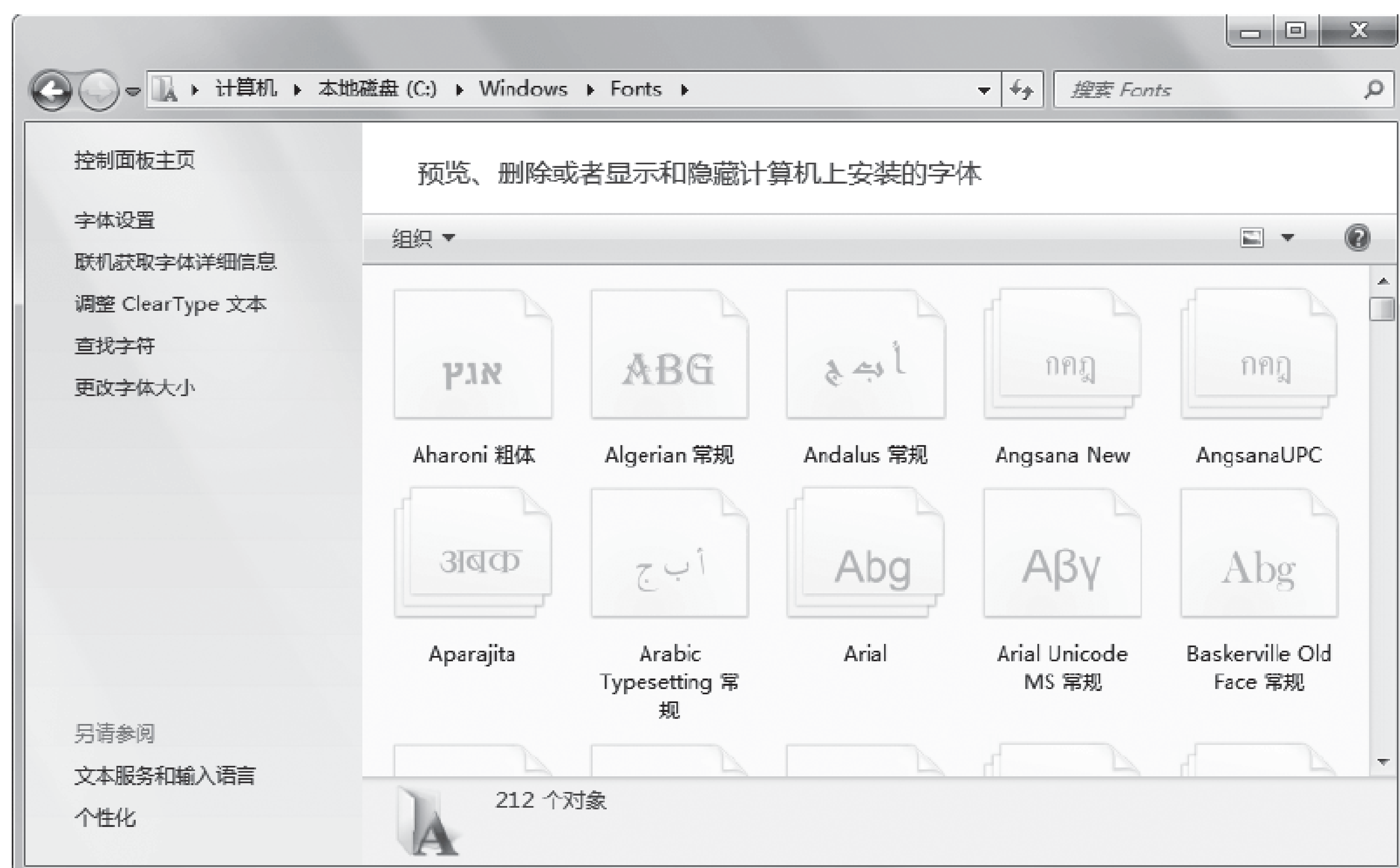
ImageDraw 模块也可以用于在图像内填写英文或中文，所使用的函数是 text( )，语法如下。

text((x,y), text, fill, font)　　　# text 是想要写入的文字

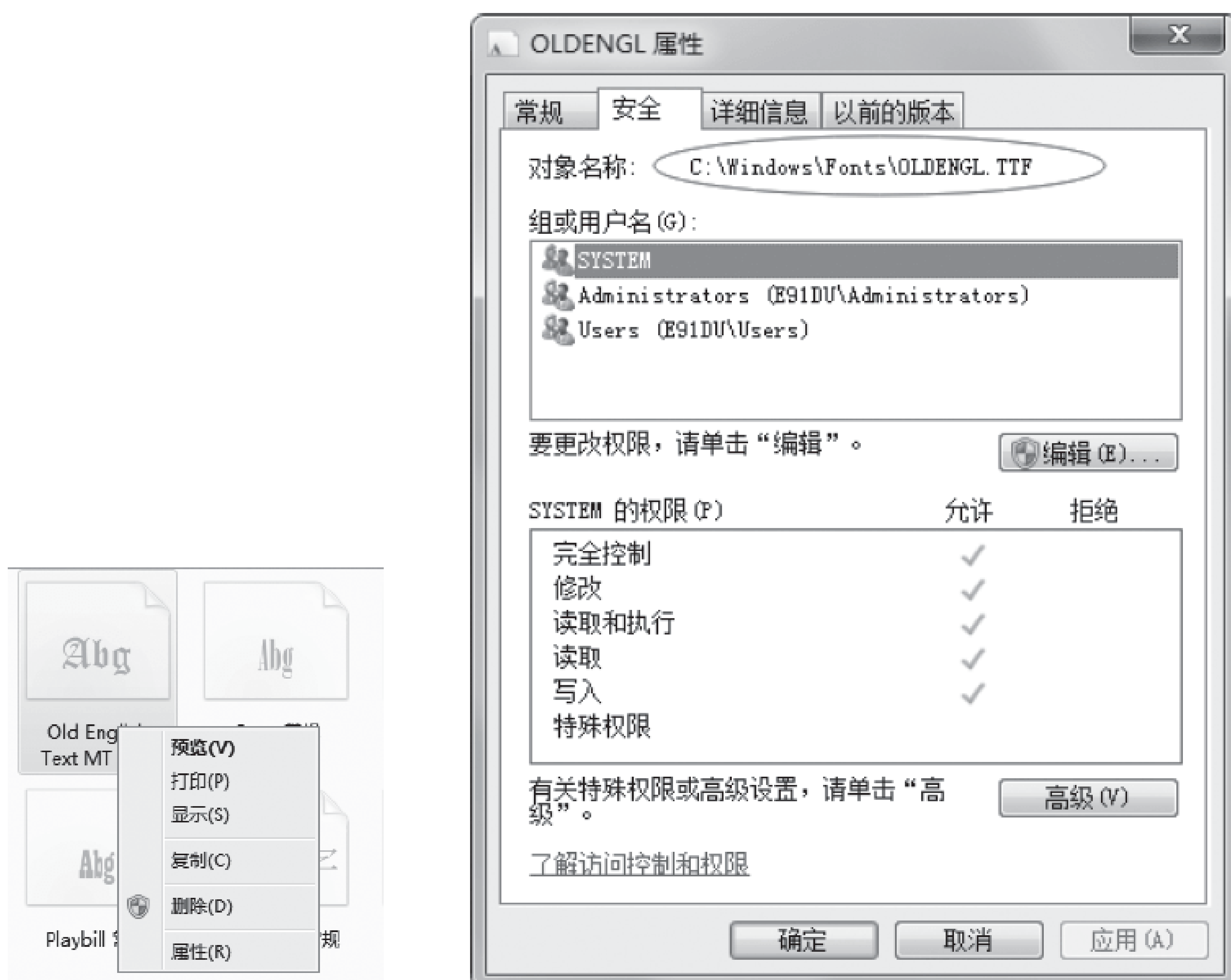
如果要使用默认方式填写文字，可以省略 font 参数，可以参考 ch17_22.py 第 8 行。如果想要使用其他字体填写文字，需调用 ImageFont.truetype( ) 方法选用字体，同时设置字号。在使用 ImageFont.truetype( ) 方法前需在程序前方导入 ImageFont 模块，可参考 ch17_22.py 第 2 行，这个方法的语法如下。

text（字体路径，字号）

在 Windows 系统中字体是放在 C:\Windows\Fonts 文件夹内，在此可以选择想要的字体。



在选定的字体文件上单击**鼠标右键**，选择“**属性**”命令，再选择“**安全**”选项卡可以看到此字体的文件名。下列是选择 Old English Text 的示范输出。



读者可以用复制的方式获得字体的路径。有了字体路径后，就可以轻松地在图像内输出各种字体了。

程序实例 ch17_22.py：在图像内填写文字，第 8、9 行是使用默认字体，执行英文字符串 Ming-Chi Institute of Technology 的输出。第 10、11 行是设置字体为 Old English Text，字号是 36，输出相同的字符串。第 13 ～ 15 行是设置字体为华康新综艺体，字号是 48，输出中文字符串“明志科技大学”。

注　如果你的计算机**没有华康新综艺体**，执行这个程序会有**错误**，所以这里附有一个 ch17_22_1.py 是使用 Microsoft 的**新细明体**，读者可以自行体会中文的输出。

```
1  # ch17_22.py
2  from PIL import Image, ImageDraw, ImageFont
3
4  newImage = Image.new('RGBA', (600, 300), 'Yellow')  # 建立300×300黄色底的图像
5  drawObj = ImageDraw.Draw(newImage)
6
7  strText = 'Ming-Chi Institute of Technology'        # 设置要打印英文字符串
8  drawObj.text((50,50), strText, fill='Blue')         # 使用默认字体与字号
9  # 使用古老英文字体, 字号是36
10 fontInfo = ImageFont.truetype('C:\Windows\Fonts\OLDENGL.TTF', 36)
11 drawObj.text((50,100), strText, fill='Blue', font=fontInfo)
12 # 处理中文字体
13 strCtext = '明志科技大学'                              # 设置要打印中文字符串
14 fontInfo = ImageFont.truetype('C:\Windows\Fonts\DFZongYiStd-W9.otf', 48)
15 drawObj.text((50,180), strCtext, fill='Blue', font=fontInfo)
16 newImage.save("out17_22.png")
```

执行结果



17-9 专题——建立 QR code/ 辨识车牌与建立停车场管理系统

17-9-1 建立 QR code

QR code 是目前最流行的**二维扫描码**，1994 年由日本 Denso-Wave 公司发明。英文字中 QR 所代表的意义是 Quick Response，意义是快速反应。QR code 最早是汽车制造商用于追踪零件，目前已应用于各行各业。它的最大特点是可以存储比普通条形码更多的资料，同时也无须对准扫描仪。

1. QR code 的应用

下列是常见的 QR code 应用：

❑ 显示网址信息

使用手机扫描可以进入此 QR code 的网址。

❑ 移动支付

消费者扫描店家的 QR code 即可完成支付。或是店家扫描消费者手机的 QR code 也可以完成支付。部分地区的停车场，也是采用司机扫描出口的 QR code 完成停车费用支付。

❑ 电子票券

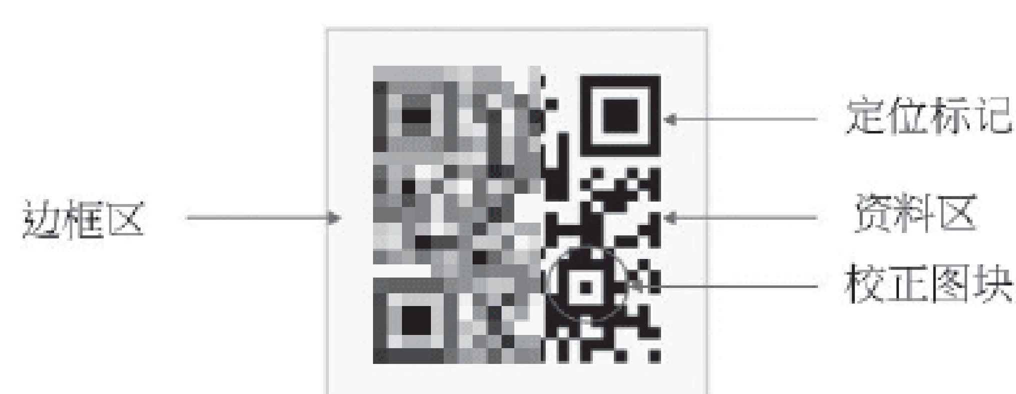
参展票、高铁票、电影票等消费者购买的票券信息，都可以使用 QR code 传输给消费者的手机，只要出示此 QR code，就可以进场了。

- 文字信息

QR code 可以存储的信息很多，常看到有的人名片上有 QR code，当扫描后就可以获得该名片主人的信息，如姓名、电话号码、地址、电子邮箱等。

2. QR code 的结构

QR code 由边框区和数据区所组成，数据区由定位标记、校正图块、版本信息、原始信息、容错信息所组成，这些信息经过编码后产生二进制字符串，白色格子代表 0，黑色格子代表 1，这些格子一般又称作模块。其实经过编码后，还会使用屏蔽（masking）方法将原始二进制字符串与屏蔽图案（Mask Pattern）做 XOR 运算，产生实际的编码，经过处理后的 QR code 辨识率将更高。QR code 基本外观如下：



- 边框区

也可以称为非数据区，主要是避免 QR code 周围的图像影响辨识。

- 定位标记

在上述图片中，左上、左下、右上是定位标记，外型类似“回”字，在使用 QR code 扫描时我们可以发现不用完全对准也可以，主要是这 3 个定位标记在帮助扫描定位。

- 校正图块

主要用于校正辨识。

- 容错修功能

QR code 有容错功能，所以如果 QR code 有破损，有时仍然可以读取，一般 QR code 的面积越大，容错能力越强。

级别	容错率
L 等级	7% 的字码可以修正
M 等级	15% 的字码可以修正
Q 等级	25% 的字码可以修正
H 等级	30% 的字码可以修正

3. QR code 的容量

QR code 目前有 40 个不同版本，版本 1 是 21×21 个模块。模块是 QR code 最小的单位，每增加一个版本，长宽各增加 4 个模块，所以版本 40 是由 177×177 个模块组成，下列是以版本 40 为例做容量解说。

资料类型	数据容量
数字	最多 7 089 个字符
字母	最多 4 296 个字符
二进制数	最多 2 953 个字符
日文汉字 / 片假名	最多 1 817 个字符（采用 Shift JIS）
中文汉字	最多 984 个字符（utf-8），最多 1800 个字符（big5/gb2312）

4. 建立 QR code 的基本知识

使用前需安装模块：

```
pip install qrcode
```

常用的几个方法如下：

```
img = qrcode.make ( " 网址数据 " )          # 产生网址数据的 QR code 对象 img
img.save ( "filename" )                    # filename 是储存 QR code 的文件名
```

程序实例 ch17_23.py：建立 http://www.deepstone.com.tw 的 QR code，这个程序会先列出 img 对象的数据形态，同时将此对象存入 out17_23.jpg 内。

```
1  # ch17_23.py
2  import qrcode
3
4  codeText = 'http://www.deepstone.com.tw'
5  img = qrcode.make(codeText)                       # 建立QR code 对象
6  print("文件格式", type(img))
7  img.save("out17_23.jpg")
```

执行结果 下列分别是执行结果与 out17_23.jpg 的 QR code 结果。

```
===================== RESTART: D:\Python\ch17\ch17_23.py =====================
文件格式 <class 'qrcode.image.pil.PilImage'>
```



程序实例 ch17_23_1.py：建立“Python 王者归来”字符串的 QR code。

```
1  # ch17_23_1.py
2  import qrcode
3
4  codeText = 'Python王者归来'
5  img = qrcode.make(codeText)                       # 建立QR code 对象
6  print("文件格式", type(img))
7  img.save("out17_23_1.jpg")
```

执行结果 扫描后可以得到下方右图的字符串。



已扫描到以下内容

Python王者归来

5. 细看 qrcode.make( ) 方法

上述我们使用 qrcode.make( ) 方法建立 QR code，这是使用预设方法建立 QR code，实际 qrcode.make( ) 方法内含 3 个子方法，整个方法原始码如下：

```
def make ( data=None, **kwargs ) :
    qr =qrcode. QRCode ( **kwargs )          # 设置条形码格式
    qr.add_data ( data )                     # 设置条形码内容
    return qr.make_image ( )                 # 建立条形码图片
```

- ❑ 设置条形码格式

它的内容如下：

qr = qrcode.QRCode(version, error_correction, box_size, border, image_factory,mask_pattern)

下列是此参数解说：

version：QR code 的版次，可以设置 1 ～ 40 的版次。

error_correction：容错率，可选 7%、15%、25%、30%，参数如下：

qrcode.constants.ERROR_CORRECT_L：7%

qrcode.constants.ERROR_CORRECT_M：15%（预设）

qrcode.constants.ERROR_CORRECT_Q：25%

qrcode.constants.ERROR_CORRECT_H：30%

box_size：每个模块的像素个数。

border：边框区的厚度，预设是 4。

image_factory：图片格式，默认是 PIL。

mask_pattern：mask_pattern 参数是 0 ～ 7，如果省略会自行使用最适当的方法。

- ❑ 设置条形码内容

qr.add_data(data)　　　　　　　# data 是所设置的条形码内容

- ❑ 建立条形码图片

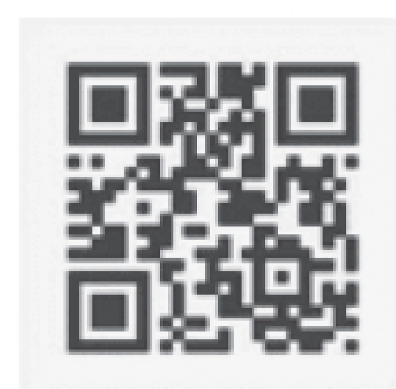
img = qr.make_image([fill_color], [back_color], [image_factory])

预设前景是黑色，背景是白色，可以使用 fill_color 和 back_color 分别更改前景和背景颜色，最后建立 qrcode.image.pil.PilImage。

程序实例 ch17_23_2.py：建立“明志科技大学”黄底蓝字的 QR code。

```
1  # ch17_23_2.py
2  import qrcode
3
4  qr = qrcode.QRCode(version=1,
5                     error_correction=qrcode.constants.ERROR_CORRECT_M,
6                     box_size=10,
7                     border=4)
8  qr.add_data("明志科技大学")
9  img = qr.make_image(fill_color='blue', back_color='yellow')
10 img.save("out17_23_2.jpg")
```

执行结果 扫描后可以得到下方右图的字符串。



已扫描到以下内容

明志科技大学

6. QR code 内有图案

有时可以看到建立 QR code 时在中央位置有图案，扫描时仍然可以获得正确的结果，这是因为 QR code 有容错能力。其实我们可以使用 17-5-3 节图像合成的方法处理。

程序实例 ch17_23_3.py：笔者将自己的图像当做 QR code 的图案，然后不影响扫描结果。在这个实例中，笔者使用蓝色白底的 QR code，同时使用 version=5。

```
1  # ch17_23_3.py
2  import qrcode
3  from PIL import Image
4  
5  qr = qrcode.QRCode(version=5,
6                     error_correction=qrcode.constants.ERROR_CORRECT_M,
7                     box_size=10,
8                     border=4)
9  qr.add_data("明志科技大学")
10 img = qr.make_image(fill_color='blue')
11 width, height = img.size              # QR code的宽与高
12 with Image.open('jhung.jpg') as obj:
13     obj_width, obj_height = obj.size
14     img.paste(obj, ((width-obj_width)//2, (height-obj_height)//2))
15 img.save("out17_23_3.jpg")
```

执行结果 读者可以自行扫描然后得到正确的结果。



7. 建立含 QR code 的名片

有时候可以看到有些人的名片上有 QR code，使用手机扫描后便能得到此名片的信息。为了完成此工作，我们必须使用 vCard（virtual card）格式。它的数据格式如下：

BEGIN:VCARD

…

特定属性数据

…

END:VCARD

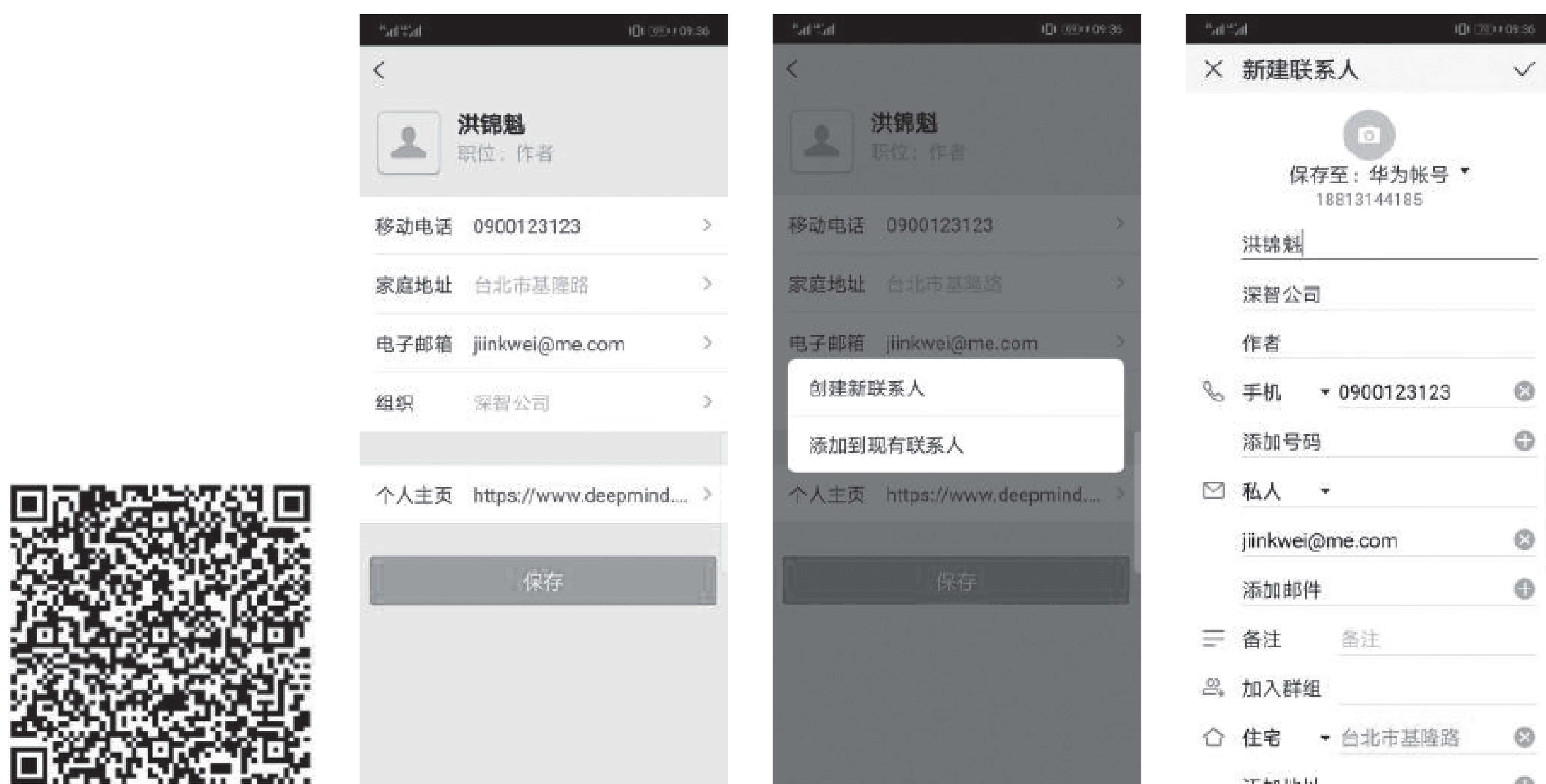
上述数据必须建在一个字符串上，未来只要将此字符串当作 QR code 数据即可。下列是常用的属性：

属性	使用说明	实例
FN	名字	FN：洪锦魁
ORG	公司抬头	ORG：深智公司
TITLE	职务名称	TITLE：作者
TEL	电话：类型 CELL：手机号 FAX：传真号	TEL：CELL：0900123123 TEL：WORK：02-22223333
	HOME：家庭号 WORK：公司号	
ADR	公司地址	ADR: 台北市基隆路
EMAIL	电子邮箱	EMAIL:jiinkwei@me.com
URL	公司网址	URL:https://www.deepmind.com.tw

程序实例 ch17_23_4.py：建立个人名片信息。

```
1  # ch17_23_4.py
2  import qrcode
3
4  vc_str = '''
5  BEGIN:VCARD
6  FN:洪锦魁
7  TEL;CELL:0900123123
8  TEL;FAX:02-27320553
9  ORG:深智公司
10 TITLE:作者
11 EMAIL:jiinkwei@me.com
12 URL:https://www.deepmind.com.tw
13 ADR:台北市基隆路
14 END:VCARD
15 '''
16
17 img = qrcode.make(vc_str)
18 img.save("out17_23_4.jpg")
```

执行结果　下列是此程序产生的 QR code。



如果读者使用微信扫描，可以读取所建立的 VCARD 资料，如果按“保存”可以列出使用何种方式保存此数据，笔者选择“创建新联系人”时，可以得到上方最右图的结果。

17-9-2　文字辨识与停车场管理系统

Tesseract-OCR 是一个**文字辨识**（Optical Character Recognition，OCR）系统，可以在多个平台上运行。目前这是一个开放资源的免费软件，1985—1994 年由惠普（HP）实验室开发，1996 年开发为适用 Windows 系统版本。有接近十年时间，这个软件没有太大进展，2005 年惠普公司将这个软件作为免费使用（open source），2006 年起这个软件改由 Google 赞助与维护。

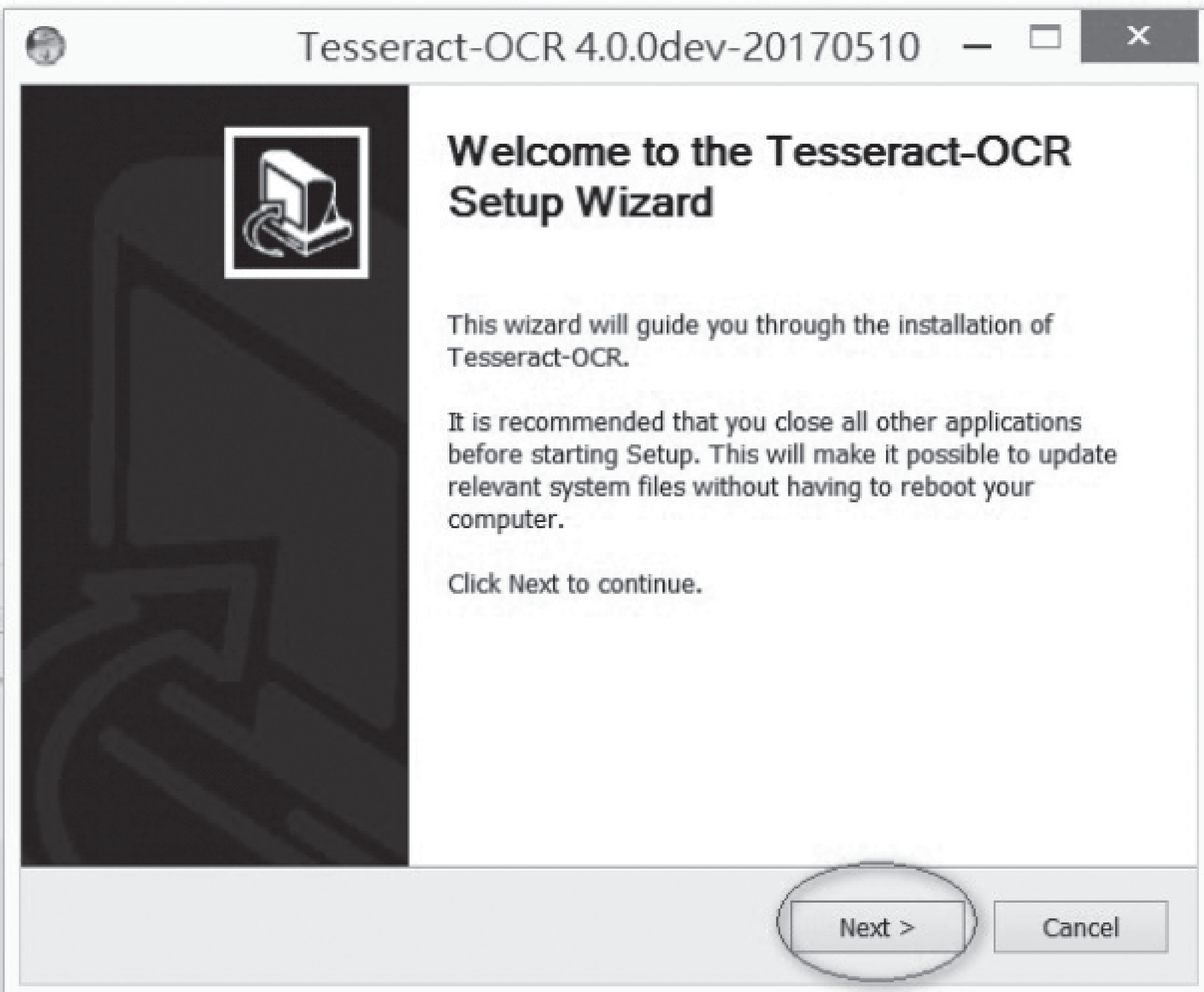
本节将简单介绍如何使用 Python 处理文字辨识，特别是应用于车牌的辨识，同时设计简单的停车管理系统。

1. 安装 Tesseract-OCR

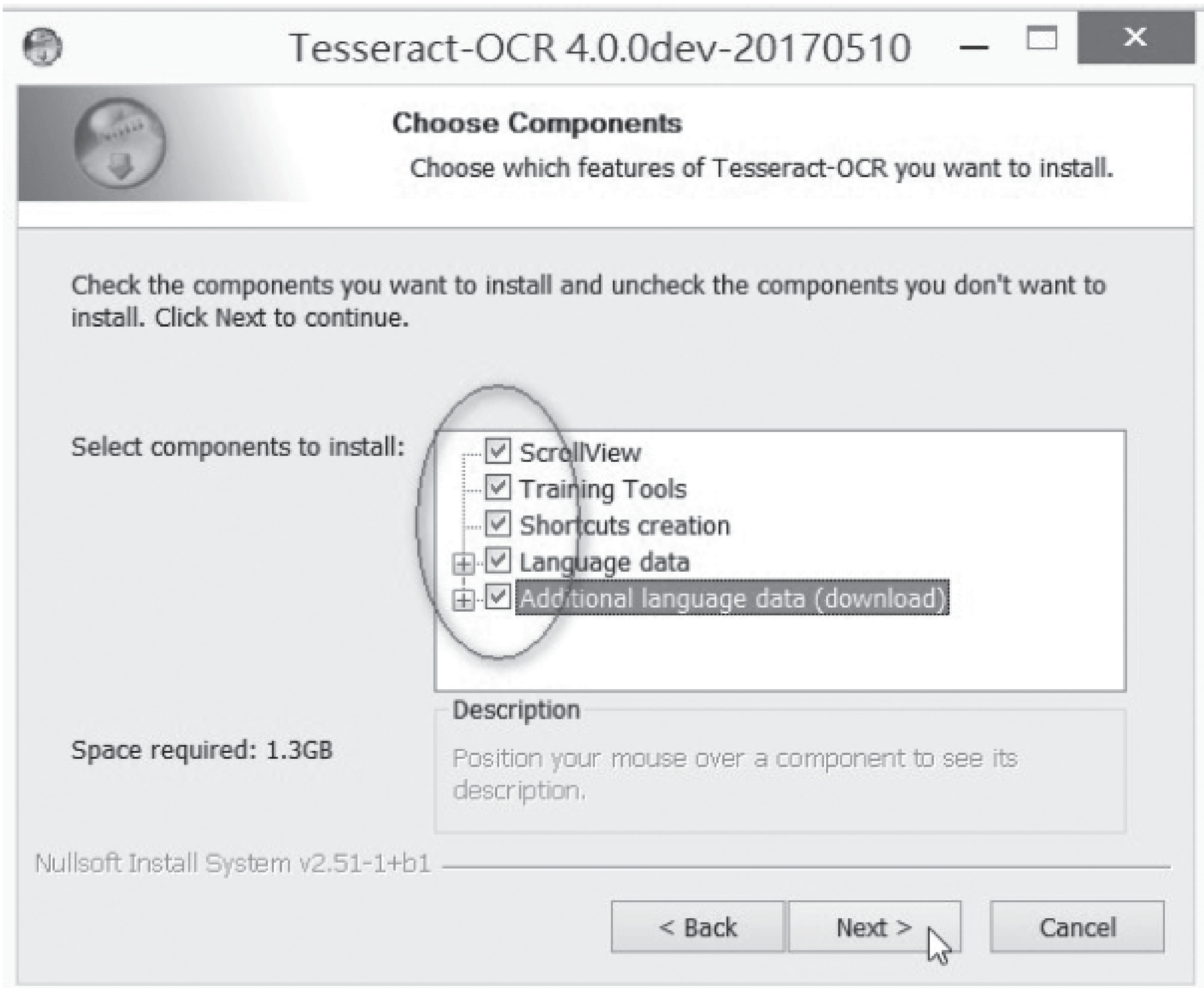
这套软件需要下载，请至下列网站。

http://digi.bib.uni-mannheim.de/tesseract/tesseract-ocr-setup-4.00.00dev.exe

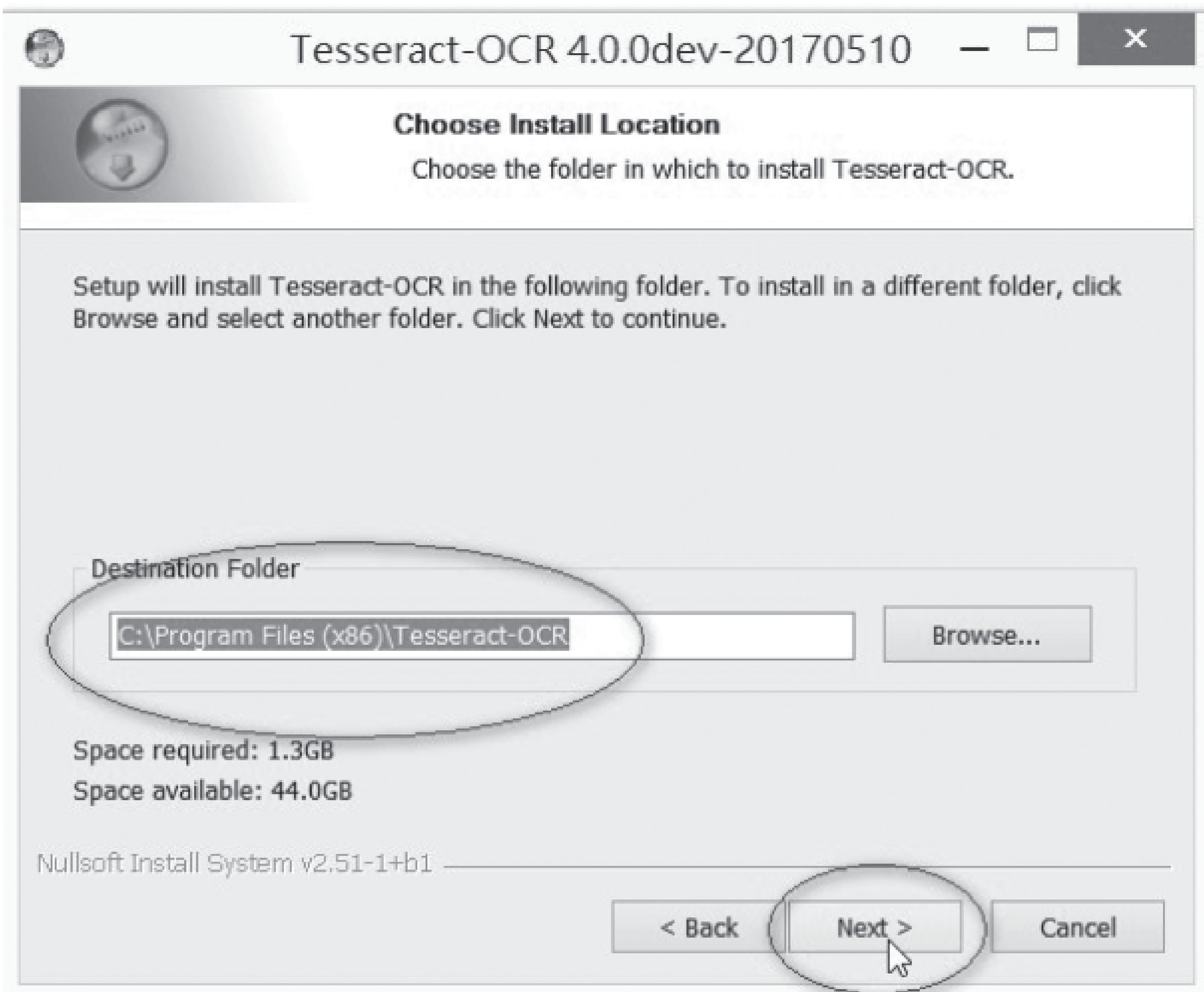
首先将看到下列画面。



单击 Next 按钮，于第 4 个画面将看到如下界面。



请勾选全部复选框，然后单击 Next 按钮。



请使用**默认目录**安装，单击 Next 按钮，接着可以使用默认设置，即可完成安装。安装完成

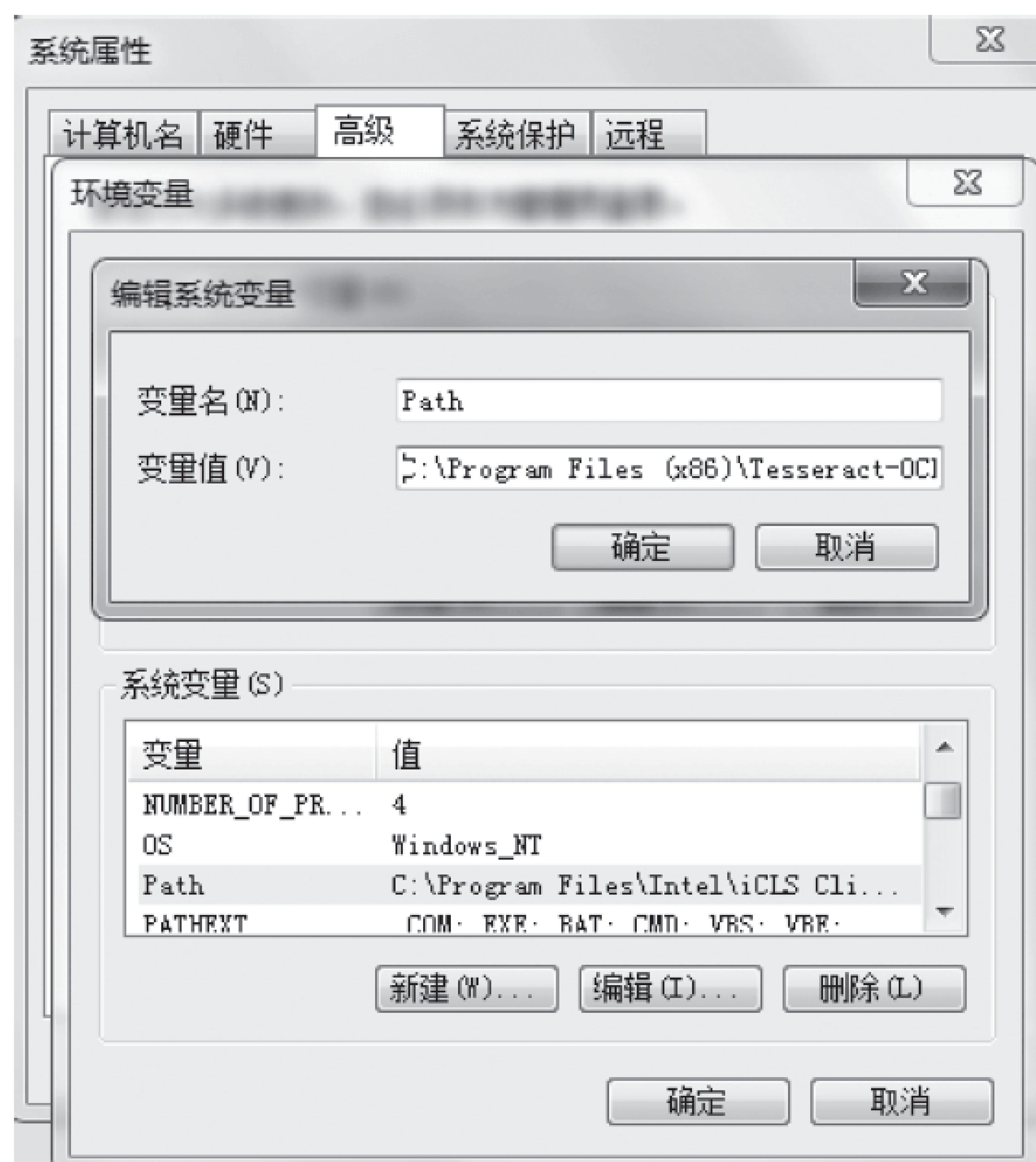
后，下一步是将 Tesseract-OCR 所在的目录设置在 Windows 操作系统的 path 路径内，这样就不会有找不到文件的问题了。首先打开控制面板中的“系统”窗口。



单击“高级系统设置”，在“**高级**”对话框中单击“**环境变量**”按钮，在“**系统变量**”栏单击 Path，会出现“**编辑系统变量**”对话框，请在“**变量值**”字段输入所安装 Tesseract 的安装目录，如果是依照默认模式输入，路径如下：

```
C:\Program Files (x86)\Tesseract-OCR
```

上述路径建议用复制方式处理，需留意不同路径的设置彼此以“;”隔开。



完成后，单击“**确定**”按钮。如果想要确定是否安装成功，可以在命令提示字符窗口输入 tesseract –v，如果有列出版本信息，就表示设置成功了。

```
C:\Users\Jiin-Kwei>tesseract -v
tesseract 4.00.00alpha
 leptonica-1.74.1
  libgif 4.1.6(?) : libjpeg 8d (libjpeg-turbo 1.5.0) : libpng 1.6.20 : libtiff 4
.0.6 : zlib 1.2.8 : libwebp 0.4.3 : libopenjp2 2.1.0


C:\Users\Jiin-Kwei>
微軟注音 半 ：
```

2. 安装 pytesseract 模块

pytesseract 是一个 Python 与 Tesseract-OCR 之间的**接口程序**，这个程序的官网就自称是 Tesseract-OCR 的 wrapper，它会自行调用 Tesseract-OCR 的内部程序执行辨识功能。调用 pytesseract 的方法，就可以完成辨识工作，可以使用下列方式安装这个模块。

```
pip install pytesseract
```

3. 文字辨识程序设计

安装完 Tesseract-OCR 后，默认情况下是可以执行英文和阿拉伯数字的辨识，下面是笔者列举了数字与英文的文件执行辨识，并将结果打印和存储，在使用 pytesseract 前，需要导入 pytesseract 模块。

```
import pytesseract
```

程序实例 ch17_24.py：这个程序会辨识车牌，所使用的车牌文件 atq9305.jpg 如下。




```
1  # ch17_24.py
2  from PIL import Image
3  import pytesseract
4
5  text = pytesseract.image_to_string(Image.open('d:\\Python\\ch17\\atq9305.jpg'))
6  print(type(text), "    ", text)
```

执行结果 这个程序无法在 Python 的 IDLE 下执行，需在命令提示环境下执行。

```
PS C:\Users\User> python d:\Python\ch17\ch17_24.py
<class 'str'>      ATQ9305
```

注：如果车牌拍的角度不好，有可能会造成辨识错误。

程序实例 ch17_25.py：这个程序会辨识车牌，同时列出车子进场时间和出场时间。如果是初次进入车辆，程序会列出车辆进场时间，同时将此车辆与进场时间用 carDict 字典存储。如果车辆已经入场，再次扫描时，系统会输出车号和此车的出场时间。

```
1  # ch17_25.py
2  from PIL import Image
3  import pytesseract
4  import time
5
6  carDict = {}
7  myPath = "d:\\Python\\ch17\\"
8  while True:
9      carPlate = input("请扫描或输入车牌(Q/q代表结束) : ")
10     if carPlate == 'Q' or carPlate == 'q':
11         break
12     carPlate = myPath + carPlate
13     keyText = pytesseract.image_to_string(Image.open(carPlate))
14     if keyText in carDict:
15         exitTime = time.asctime()
16         print("车辆出场时间 : ", keyText, ":", exitTime)
17         del carDict[keyText]
18     else:
19         entryTime = time.asctime()
20         print("车辆入场时间 : ", keyText, ":", entryTime)
21         carDict[keyText] = entryTime
```

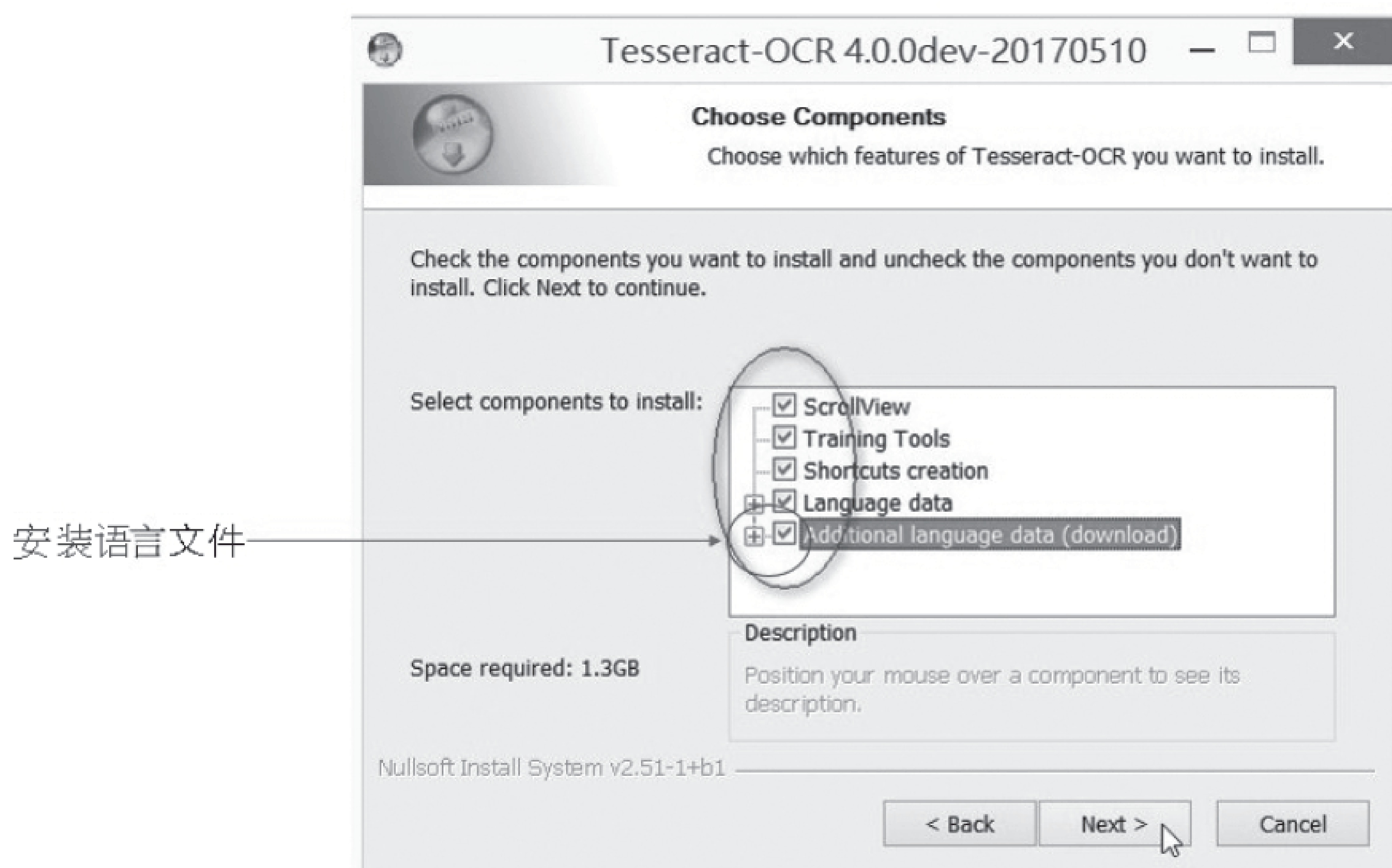
执行结果

```
PS C:\Users\User> python d:\Python\ch17\ch17_25.py
请扫描或输入车牌(Q/q代表结束) : atq9305.jpg
车辆入场时间 :  ATQ9305 : Mon Jun  3 07:30:14 2019
请扫描或输入车牌(Q/q代表结束) : rbt1388.jpg
车辆入场时间 :  RBT- 13838 : Mon Jun  3 07:30:30 2019
请扫描或输入车牌(Q/q代表结束) : atq9305.jpg
车辆出场时间 :  ATQ9305 : Mon Jun  3 07:31:41 2019
请扫描或输入车牌(Q/q代表结束) : q
PS C:\Users\User>
```

读者须留意有时辨识结果还是会有小错误。

17-9-3　辨识繁体中文

Tesseract-OCR 也可以辨识繁体中文，这时需要指示程序引用中文数据文件，这个繁体中文数据文件名称是 chi-tra.traineddata，在 17-9-2 节的安装画面中，有指出需要设置安装语言文件。



如果读者依照上面的指示安装，可以在 \tessdata 文件夹下看到 chi_tri.trianeddata 文件，下面将以实例 ch17_26.py 说明识别下列繁体中文的文件。

1：從無到有一步一步教導讀者 R 語言的使用

2：學習本書不需要有統計基礎，但在無形中本

書已灌溉了統計知識給你

程序实例 ch17_26.py：执行繁体中文图片文字的辨识，这个程序最重要的是在 image_to_string( ) 方法内增加了第 2 个参数 lang='chi_tra'，这个参数会引导程序使用繁体中文数据文件做辨识。

```
1  # ch17_26.py
2  from PIL import Image
3  import pytesseract
4
5  text  = pytesseract.image_to_string(Image.open('d:\\Python\\ch17\\data17_26.jpg'),
6                                      lang='chi_tra')
7  print(text)
8  with open('d:\\Python\\ch17\\out17_26.txt', 'w') as fn:
9      fn.write(text)
```

执行结果

```
PS C:\Users\User> python d:\Python\ch17\ch17_26.py
1 : 從 無 到 有 一 步 一 步 教 導 讀 者 R 語 言 的 使 用
2 : 學 習 本 書 不 需 要 有 統 計 基 礎 , 但 在 無 形 中 本
書 已 灌 溉 了 統 計 知 識 給 你
```

在上述辨识处理中，没有错误，这是一个非常好的辨识结果。不过在使用时发现，如果图片的字比较小，会有较多辨识错误。

17-9-4 辨识简体中文

辨识简体中文和繁体中文步骤相同，只是导入的是 chi_sim.trianeddata 简体中文文件，实例 ch17_27.py 将说明如何识别下列简体中文的图片。

1：从无到有一步一步教导读者 R 语言的使用

2：学习本书不需要有统计基础，但在无形中本

书已灌溉了统计知识给你

程序实例 ch17_27.py：执行简体中文图片文字的辨识。这个程序最重要的是在 image_to_string( ) 方法内增加了第 2 个参数 lang='chi_sim'，这个参数会引导程序使用简体中文数据文件做辨识。这个程序另外需留意的是，第 8 行在打开文件时需要增加 encoding='utf-8'，才可以将简体中文写入文件。

```
1  # ch17_27.py
2  from PIL import Image
3  import pytesseract
4
5  text  = pytesseract.image_to_string(Image.open('d:\\Python\\ch17\\data17_27.jpg'),
6                                      lang='chi_sim')
7  print(text)
8  with open('d:\\Python\\ch17\\out17_27.txt', 'w', encoding='utf-8') as fn:
9      fn.write(text)
```

执行结果

```
PS C:\Users\User> python d:\Python\ch17\ch17_27.py
1: 从 无 到 有 一 步 一 步 教 导 读 者 R 语 言 的 使 用
2: 学 习 本 书 不 需 要 有 统 计 基 础 , 但 在 无 形 中 本
书 已 灌 渥 了 统 计 知 识 给 你
```

在使用时，笔者也发现如果发生无法辨识的情况，程序将响应空白。

17-10 专题——词云 (Word Cloud) 设计

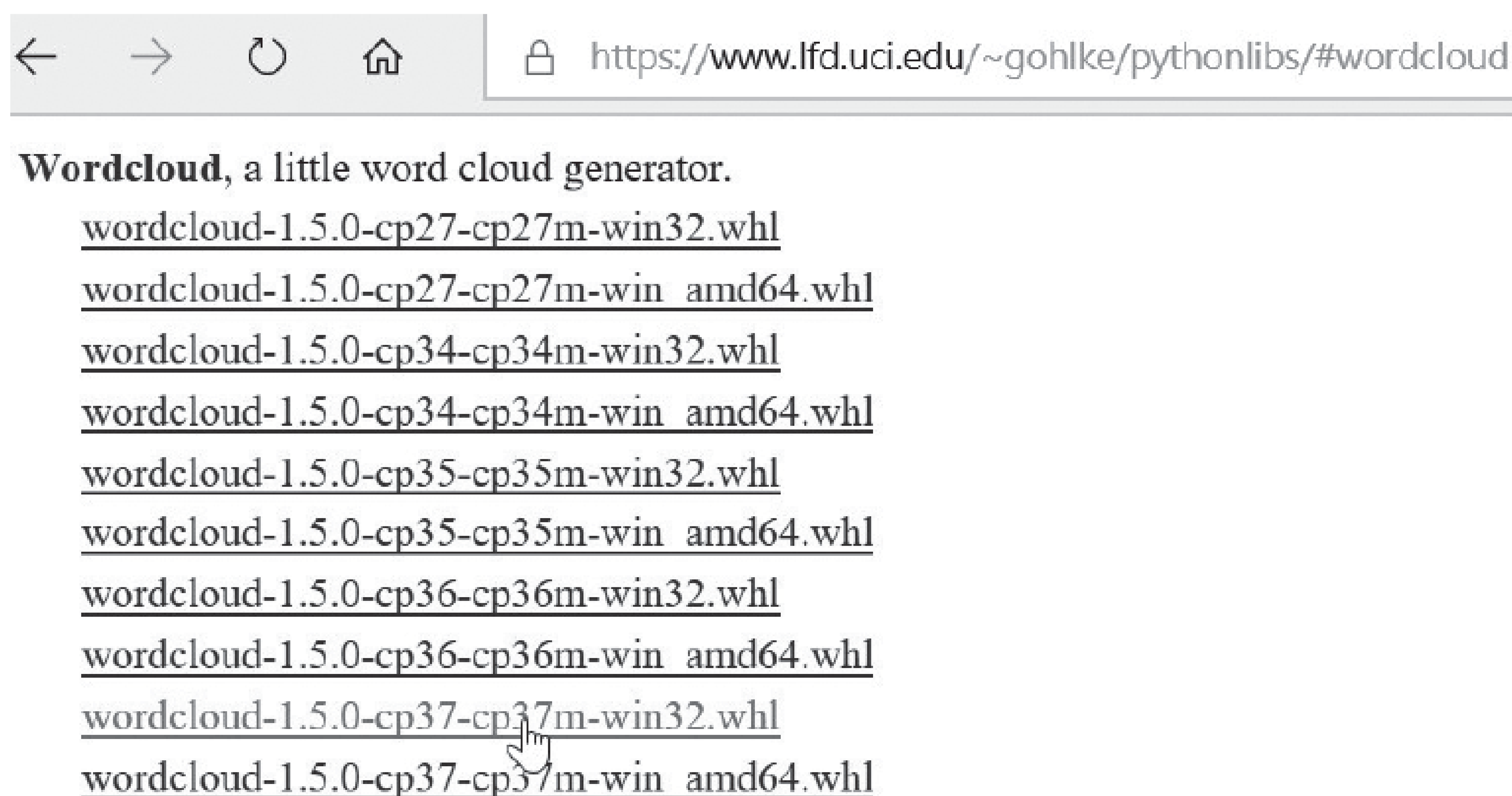
17-10-1 安装 Word Cloud

如果想建立词云（Word Cloud），首先需下载相对应的 Python 版本和硬件的 whl 文件，然后用

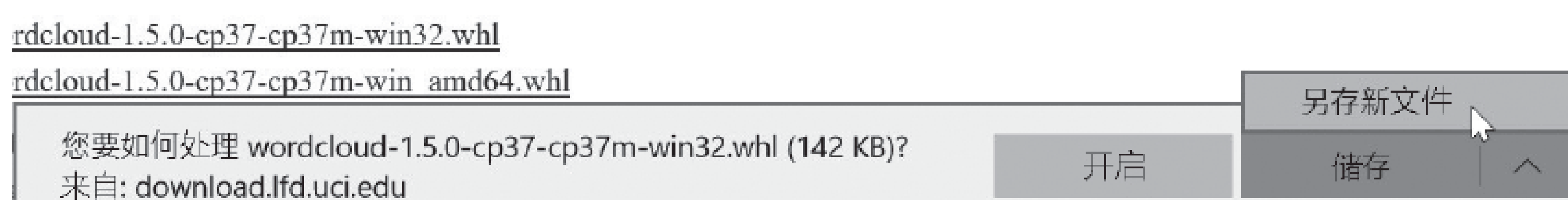
此文件安装 Wordcloud 模块，请进入下列网址：

https://www.lfd.uci.edu/~gohlke/pythonlibs/#wordcloud

然后请进入下列界面，同时选择自己目前系统环境适用的文件，此例笔者选择如下。



单击下载后，可以在窗口下方看到应如何处理此文件，请选择另存新文件，笔者此时将此文件存放在 d:\Python\ch17。



存储文件后，就可以进入 DOS 环境使用 pip install 文件，安装所下载的文件。

```
PS D:\> pip install d:\Python\ch22\wordcloud-1.5.0-cp37-cp37m-win32.whl
```

如果成功安装将可以看到下列信息。

```
Installing collected packages: wordcloud
Successfully installed wordcloud-1.5.0
```

17-10-2　我的第一个词云程序

要建立词云程序，首先是导入 Wordcloud 模块，可以使用下列语法。

```
from wordcloud import WordCloud
```

除此之外，必须为词云建立一个 txt 文本文件，未来此文件的文字将出现在词云内，下列是笔者所建立的 text17_28.txt 文件。



产生词云的步骤如下。

（1）读取词云的文本文件。

（2）词云使用 Worldcloud( )，此方法不含参数表示使用默认环境，然后使用 generate( ) 建立步

骤（1）中文本文件的词云对象。

（3）词云对象使用 to_image( ) 建立词云图像文件。

（4）使用 show( ) 显示词云图像文件。

程序实例 ch17_28.py：我的第一个词云程序。

```
1  # ch17_28.py
2  from wordcloud import WordCloud
3
4  with open("text17_28.txt") as fp:       # 英文的文本文件
5      txt = fp.read()                     # 读取文件
6
7  wd = WordCloud().generate(txt)          # 由txt文字产生WordCloud对象
8  imageCloud = wd.to_image()              # 由WordCloud对象建立词云图像文件
9  imageCloud.show()                       # 显示词云图像文件
```

执行结果



其实屏幕显示的是一个图片框文件，此例只列出词云图片，每次执行都会看到不一样字词排列的词云图片，如上方所示。上述背景预设是黑色，未来会介绍使用 background_color 参数更改背景颜色，这将是读者的习题。上述第 8 行是使用词云对象的 to_image( ) 方法产生词云图片的图像文件，第 9 行则是使用词云对象的 show( ) 方法显示词云图片。

其实也可以使用 matplotlib 模块的方法产生词云图片的图像文件，并显示词云图片的图像文件，未来会做说明。

17-10-3　建立含中文字的词云结果失败

使用程序实例 ch17_28.py，但是使用中文字的 txt 文件时，将无法正确显示词云，可参考 ch17_29.py。

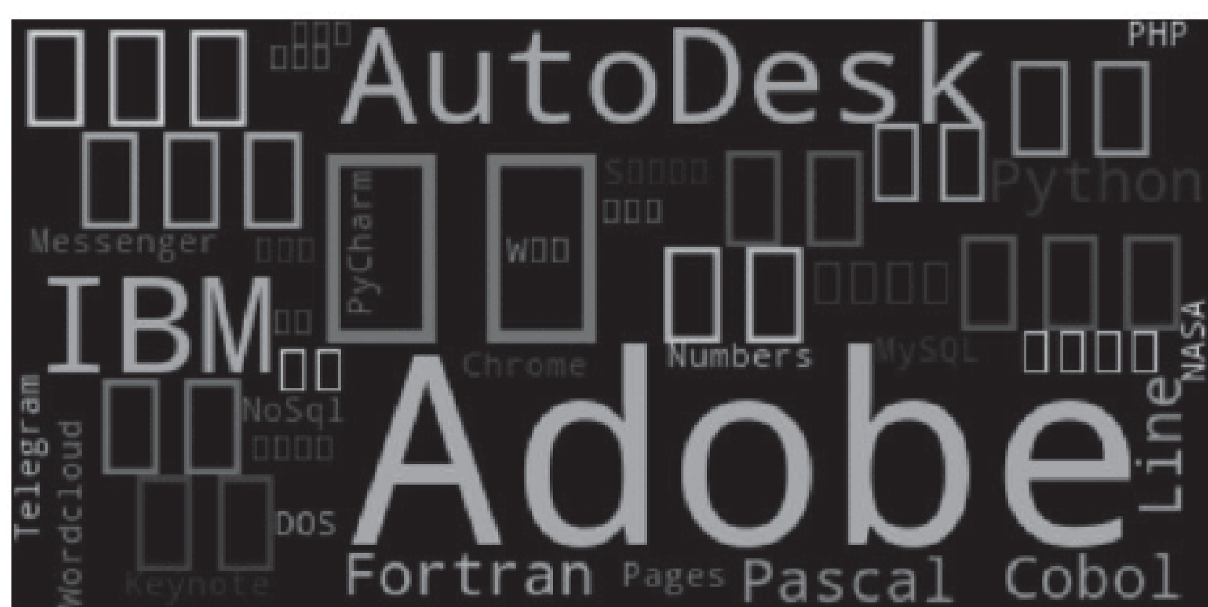
程序实例 ch17_29.py：无法正确显示中文字的词云程序，本程序的中文词云文件 utf17_29.txt 如下。

```
微软 Adobe AutoDesk IBM 谷歌 脸书 甲骨文 华硕 台积电 联电
亚马逊 宏碁 Python C C++ Pascal Fortran Cobol 汇编语言 台
微信 Line Messenger Telegram Keynote Pages Numbers Chrome
S软件银行 NASA 文魁 数据库 大数据 NoSql PHP MySQL DOS 数扌
W窗口 浏览器 PyCharm Wordcloud
```

下列是程序代码内容。

```
1  # ch17_29.py
2  from wordcloud import WordCloud
3
4  with open("utf17_29.txt", encoding='utf-8') as fp:
5      txt = fp.read()                          # 读取文件
6
7  wd = WordCloud().generate(txt)              # 由txt文字产生WordCloud对象
8  imageCloud = wd.to_image()                  # 由WordCloud对象建立词云图像文件
9  imageCloud.show()                           # 显示词云图像文件
```

执行结果



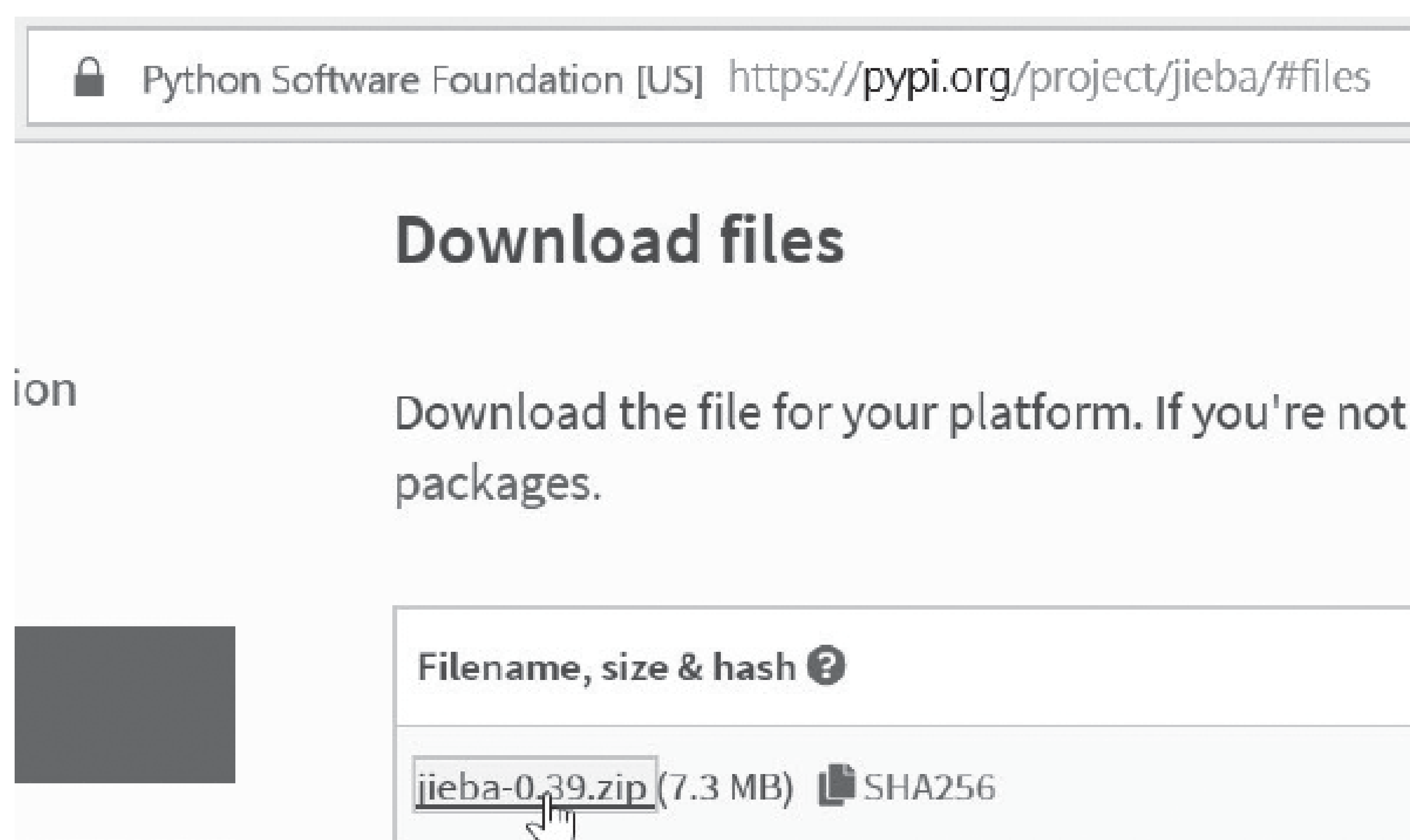
上述结果很明显，中文字无法正常显示，用方框代表。

17-10-4　建立含中文字的词云

首先需要安装中文分词函数库模块 jieba（也有人翻译为**结巴**），这个模块可以用于句子与词的分割、标注，可以进入下列网站下载。

https://pypi.org/project/jieba/#files

然后请下载 jieba-0.39.zip 文件。



下载完成后需要解压缩，笔者是将此文件存储在 d:\Python\ch17，然后进入此解压缩文件的文件夹，输入 python setup.py install 安装 jieba 模块。

```
PS D:\Python\ch17\jieba-0.39> cd jieba-0.39
PS D:\Python\ch17\jieba-0.39\jieba-0.39> python setup.py install
```

jieba 模块内有 cut( ) 方法，这个方法可以将所读取的文件执行分词，英文文件由于每个单字空一格所以比较单纯，中文文件则是借用 jieba 模块的 cut( ) 方法。由于我们希望所断的词可以空一格，所以可以采用下列语句执行。

```
cut_text = ' '.join(jieba.cut(txt))          # 产生分词的字串
```

此外，需要为词云建立对象，所采用的方法是 generate( )，整个语句如下。

```
wordcloud = WordCloud(                        # 建立词云对象
    font_path="C:/Windows/Fonts\mingliu",
    background_color="white",width=1000,height=880).generate(cut_text)
```

在上述建立含中文字的词云对象时，需要在 Worldcloud( ) 方法内增加 font_path 参数，这是设置中文字所使用的字体。另外，笔者也增加了 background_color 参数设置词云的背景颜色，width 是设置单位是像素的宽度，height 是设置单位是像素的高度，若是省略 background_color、width、height 则使用默认值。

程序实例 ch17_30.py：建立含中文字的词云图像。

```
1  # ch17_30.py
2  from wordcloud import WordCloud
3  import jieba
4
5  with open("utf17_29.txt", encoding='utf-8') as fp:
6      txt = fp.read()                          # 读取文件
7
8  cut_text = ' '.join(jieba.cut(txt))          # 产生分词的字符串
9
10 wd = WordCloud(                              # 建立词云对象
11     font_path="C:/Windows/Fonts\mingliu",
12     background_color="white",width=1000,height=880).generate(cut_text)
13
14 imageCloud = wd.to_image()                   # 由WordCloud对象建立词云图像文件
15 imageCloud.show()                            # 显示词云图像文件
```

执行结果



在建立词云图像文件时，也可以使用 matplotlib 模块，使用此模块的 imshow( ) 建立词云图像文件，然后使用 show( ) 显示词云图像文件。

程序实例 ch17_31.py：使用 matplotlib 模块建立与显示词云图像，同时将宽设为 800，高设为 600。

```
1  # ch17_31.py
2  from wordcloud import WordCloud
3  import matplotlib.pyplot as plt
4  import jieba
5
6  with open("utf17_29.txt", encoding='utf-8') as fp:
7      txt = fp.read()                        # 读取文件
8
9  cut_text = ' '.join(jieba.cut(txt))        # 产生分词的字符串
10
11 wd = WordCloud(                            # 建立词云对象
12     font_path="C:/Windows/Fonts\mingliu",
13     background_color="white",width=800,height=600).generate(cut_text)
14
15 plt.imshow(wd)                             # 由WordCloud对象建立词云图像文件
16 plt.show()                                 # 显示词云图像文件
```

执行结果



通常以 matplotlib 模块显示词云图像文件时，可以增加 axis（"off"）关闭轴线。

程序实例 ch17_32.py：关闭显示轴线，同时背景颜色改为黄色。

```
1  # ch17_32.py
2  from wordcloud import WordCloud
3  import matplotlib.pyplot as plt
4  import jieba
5
6  with open("utf17_29.txt", encoding='utf-8') as fp:
7      txt = fp.read()                        # 读取文件
8
9  cut_text = ' '.join(jieba.cut(txt))        # 产生分词的字符串
10
11 wd = WordCloud(                            # 建立词云对象
12     font_path="C:/Windows/Fonts\mingliu",
13     background_color="yellow",width=800,height=400).generate(cut_text)
14
15 plt.imshow(wd)                             # 由WordCloud对象建立词云图像文件
16 plt.axis("off")                            # 关闭显示轴线
17 plt.show()                                 # 显示词云图像文件
```

执行结果



注 中文分词是人工智能应用在中文语意分析 (semantic analysis) 的一门学问，对于英文字而言由于每个单字用空格或标点符号分开，所以可以很容易地执行分词。所有中文字之间没有空格，所以要将一段句子内有意义的词语解析，比较困难，一般是用匹配方式或统计学方法处理。目前精准度已经达到 97% 左右，细节则不在本书讨论范围。

17-10-5 进一步认识 jieba 模块的分词

前面所使用的文本文件，中文字部分均是一个公司名称的名词，文字内容又适度空一格了，我们也可以将词云应用于一整段文字，这时可以看到 jieba 模块的 cut( ) 方法自动分割整段中文的功力，其正确率高达百分之九十以上。

程序实例 ch17_33.py：将 utf17_33.txt 应用于 ch17_32.py。

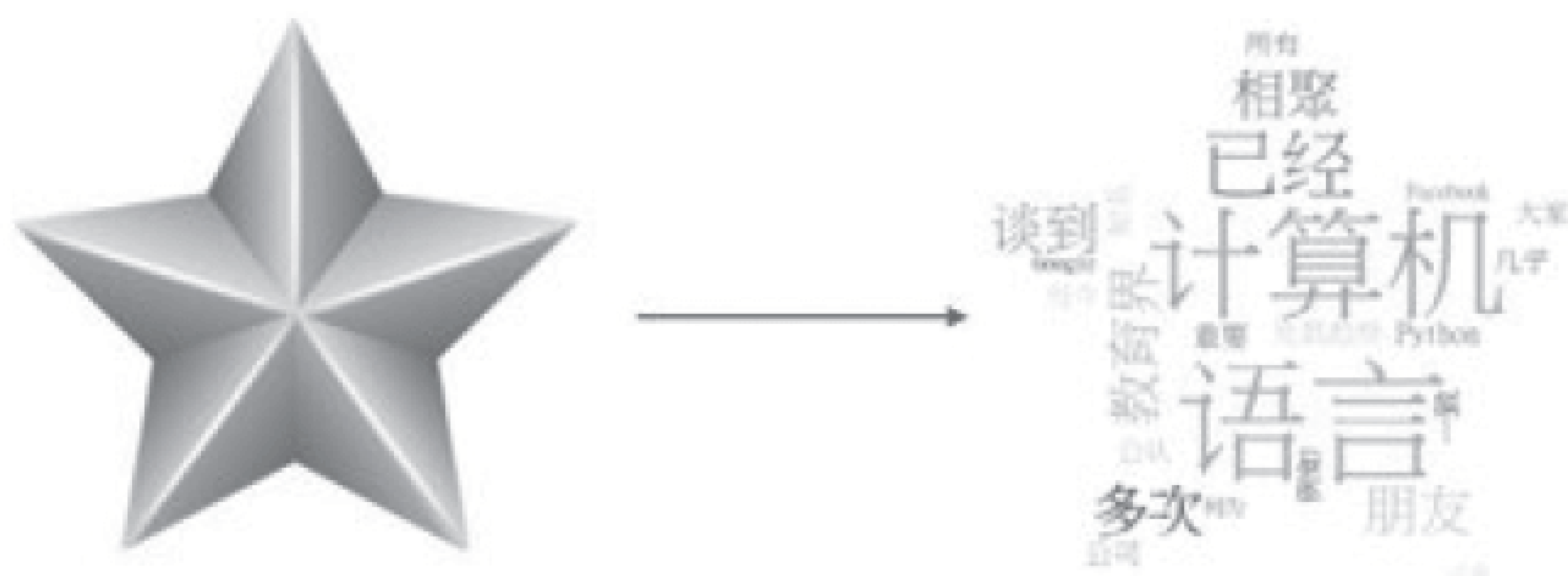
```
1  # ch17_33.py
2  from wordcloud import WordCloud
3  import matplotlib.pyplot as plt
4  import jieba
5
6  with open("utf17_33.txt", encoding='utf-8') as fp:
7      txt = fp.read()                          # 读取文件
8
9  cut_text = ' '.join(jieba.cut(txt))          # 产生分词的字符串
10
11 wd = WordCloud(                              # 建立词云对象
12     font_path="C:/Windows/Fonts\mingliu",
13     background_color="yellow",width=800,height=400).generate(cut_text)
14
15 plt.imshow(wd)                               # 由WordCloud对象建立词云图像文件
16 plt.axis("off")                              # 关闭显示轴线
17 plt.show()                                   # 显示词云图像文件
```

执行结果



17-10-6　建立含图片背景的词云

在先前所产生的词云外观是矩形，建立词云时，另一个特点是可以依据图片的外形产生词云，如果有一个无背景的图片，可以依据此图片产生相同外形的词云。



要建立这类的词云需增加使用 Numpy 模块，可参考下列语句。

```
bgimage = np.array(Image.open("star.gif"))
```

上述 np.array( ) 是建立数组所使用的参数是 Pillow 对象，这时可以将图片用大型矩阵表示，然后在有颜色的地方填词。最后在 WordCloud( ) 方法内增加 mask 参数，执行屏蔽限制图片形状，如下所示。

```
wordcloud = WordCloud(
    font_path="C:/Windows/Fonts\mingliu",
    background_color="white",
    mask=bgimage).generate(cut_text)
```

需留意当使用 mask 参数后，width 和 height 的参数设置就会失效，所以此时可以省略设置这两个参数。本程序所使用的星图 star.gif 是一个星状的无背景图。

程序实例 ch17_34.py：建立星状的词云图，所使用的背景文件是 star.gif，所使用的文本文件是 utf17_33.txt。

```
1  # ch17_34.py
2  from wordcloud import WordCloud
3  from PIL import Image
4  import matplotlib.pyplot as plt
5  import jieba
6  import numpy as np
7
8  with open("utf17_33.txt", encoding='utf-8') as fp:
9      txt = fp.read()                            # 读取文件
10 cut_text = ' '.join(jieba.cut(txt))            # 产生分词的字符串
11
12 bgimage = np.array(Image.open("star.gif"))     # 背景图
13
14 wd = WordCloud(                                # 建立词云对象
15     font_path="C:/Windows/Fonts\mingliu",
16     background_color="white",
17     mask=bgimage).generate(cut_text)           # mask设置
18
19 plt.imshow(wd)                                 # 由WordCloud对象建立词云图像文件
20 plt.axis("off")                                # 关闭显示轴线
21 plt.show()                                     # 显示词云图像文件
```

执行结果

程序实例 ch17_35.py：建立人形的词云图，所使用的背景文件是 hung.gif，所使用的文本文件是 text17_28.txt，所使用的字体是 C:\Windows\Fonts\OLDENGL.Tif。

```
1   # ch17_35.py
2   from wordcloud import WordCloud
3   from PIL import Image
4   import matplotlib.pyplot as plt
5   import numpy as np
6
7   with open("text17_28.txt") as fp:               # 含中文的文本文件
8       txt = fp.read()                             # 读取文件
9
10  bgimage = np.array(Image.open("hung.gif"))      # 背景图
11
12  wd = WordCloud(                                 # 建立词云对象
13      font_path="C:/Windows/Fonts\OLDENGL.TTF",
14      background_color="white",
15      mask=bgimage).generate(txt)                 # mask设置
16
17  plt.imshow(wd)                                  # 由WordCloud对象建立词云图像文件
18  plt.axis("off")                                 # 关闭显示轴线
19  plt.show()                                      # 显示词云图像文件
```

执行结果

hung.gif

习题

1. 请用自己的大头照，使用更改宽度与高度的方法，重设大小，需留意宽度与高度必须是整数，必须附上正常和其他 8 种变化，变化方式如下。(17-4 节)

（1）高度不变，宽度是 1.2 倍。

（2）高度不变，宽度是 1.5 倍。

（3）高度不变，宽度是 0.5 倍。

（4）高度不变，宽度是 0.8 倍。

（5）宽度不变，高度是 1.2 倍。

（6）宽度不变，高度是 1.5 倍。

（7）宽度不变，高度是 0.8 倍。

（8）宽度不变，高度是 0.5 倍。

下列是 Python Shell 窗口中的执行结果，与文件夹内的文件结果。

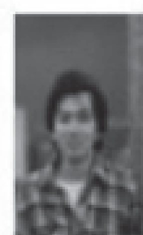
```
====================== RESTART: D:/Python/ex/ex17_1.py ======================
>>>
```

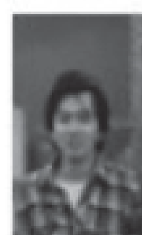

hung


out17_1


out17_2


out17_3


out17_4

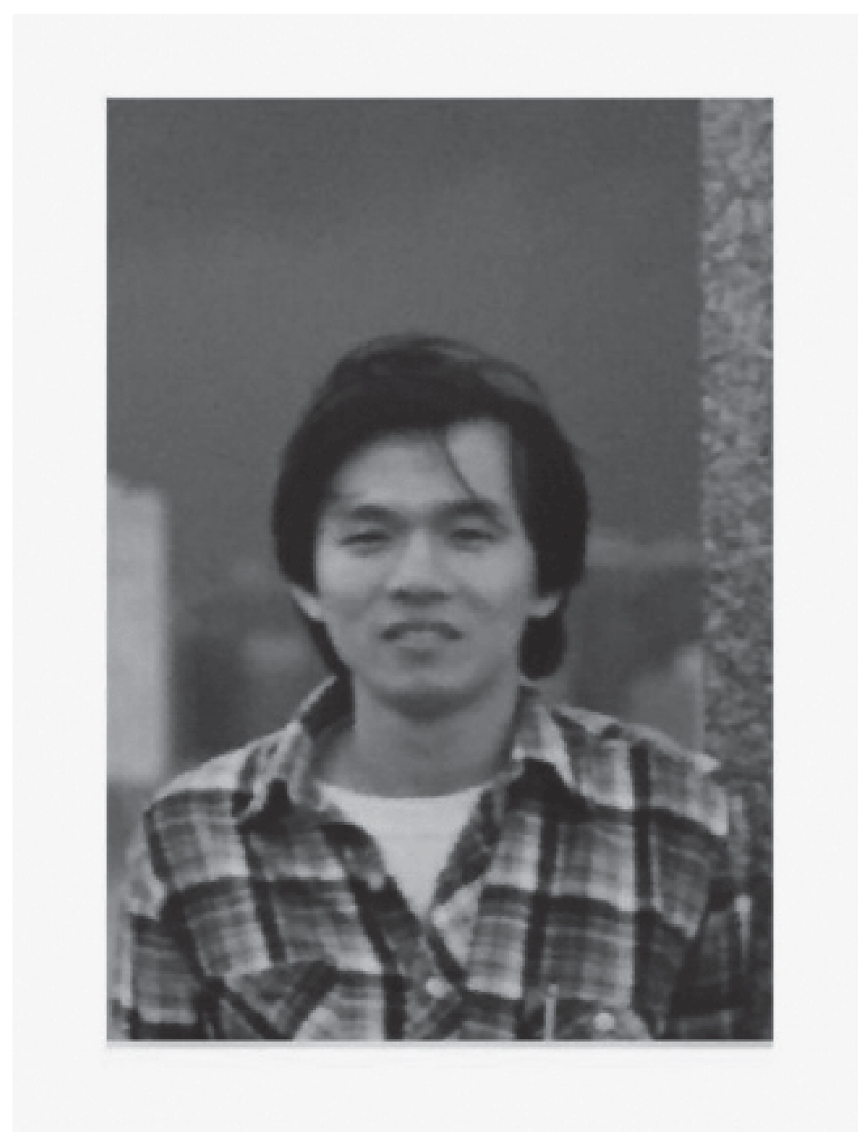

out17_5


out17_6


out17_7


out17_8

2. 请用自己的大头贴照片，将此照片的大小改为 350×500，然后在此照片四周增加 50 的外框，将执行结果存入 fig17_2.jpg。(17-5 节)



3. 请参考护照照片规格，将自己的大头贴参考 17_18.py 方式布局在图像文件内。护照相片大小是 3.5cm×4.5cm，若分辨率是 72 像素 / 英寸，则像素是 99×127。(17-5 节)



4. 请参考 ch17_19.py，但是所使用的相片是自行拍摄的学校风景，请参考 17-6 节的 10 种滤镜特效处理，然后列出结果，下列图片是参考。(17-6 节)


fig17_4_BLUR


fig17_4_CONTOUR


fig17_4_DETAIL


fig17_4_EDGE_ENHANCE


fig17_4_EDGE_ENHANCE_MORE


fig17_4_EMBOSS

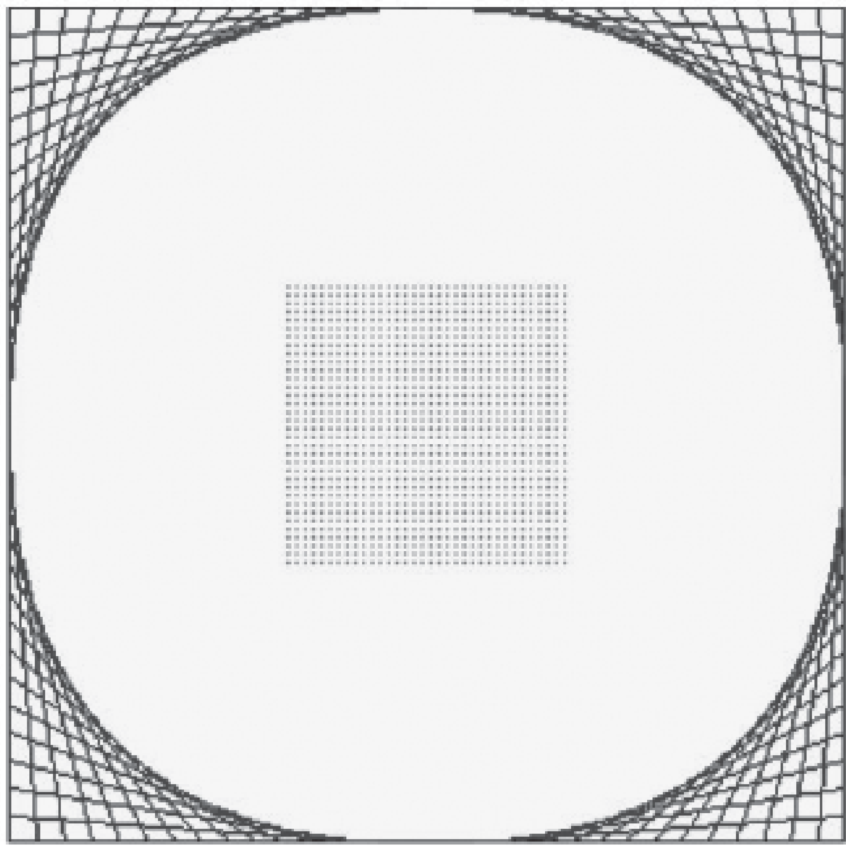

fig17_4_FIND_EDGES


fig17_4_SHARPEN

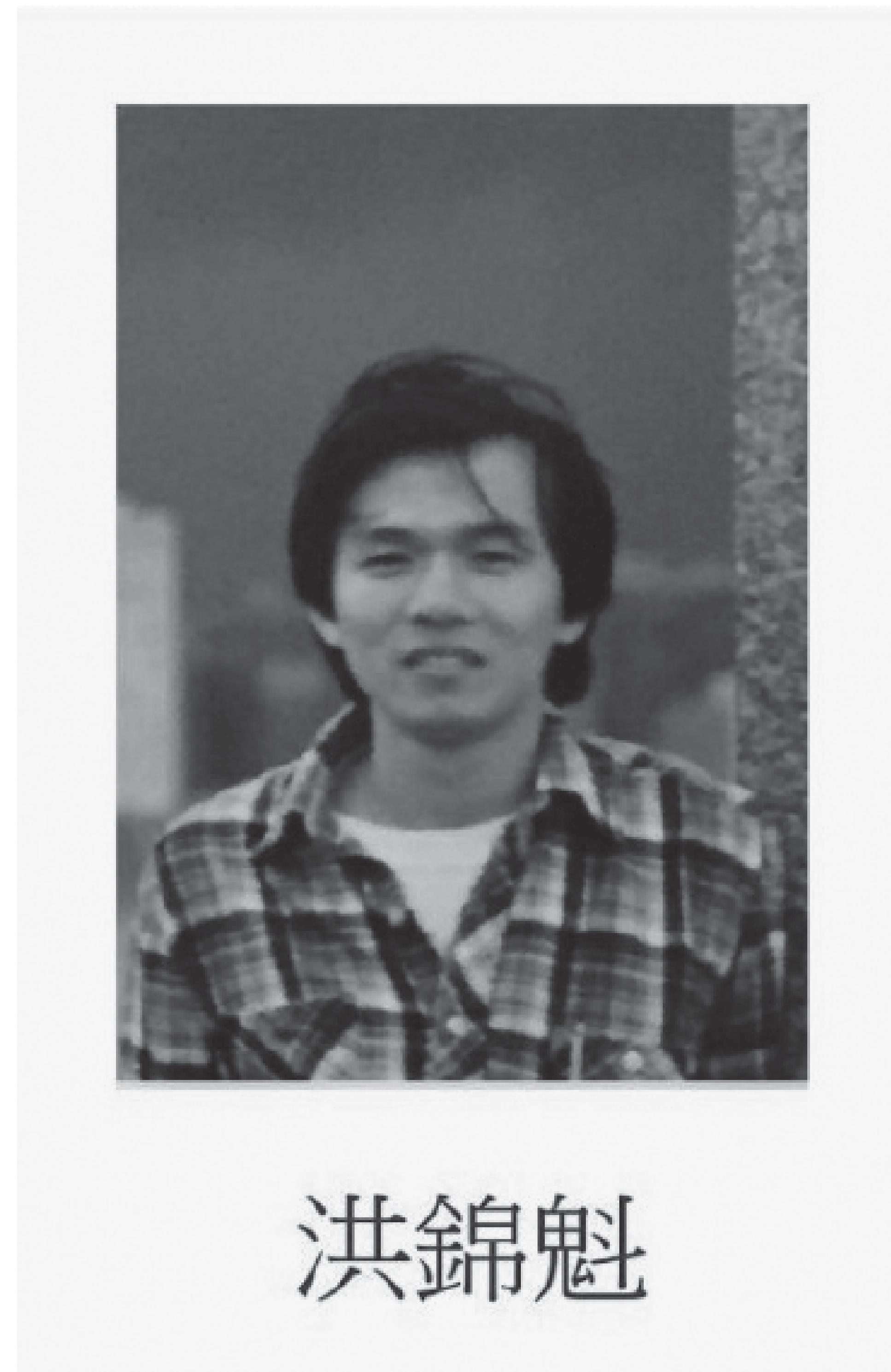

fig17_4_SMOOTH


fig17_4_SMOOTH_MORE

5. 请参考 ch17_20.py，扩充此程序功能，将美工线条的概念应用在左上角与右下角。(17-7 节)



6. 请用自己的大头贴照片，将此照片的大小改为 350×500，然后在此照片的上、左、右增加 50 的外框，下方则增加 200 的外框，然后将执行结果存入 fig17_6.jpg，最后在下方填入自己的名字。(17-8 节)



7. 请建立自己母校的 QR code。(17-9 节)

8. 扩充设计 ch17_25.py，假设每小时收费是 60 元，不足一小时以一小时收费。出场时会列出停车费用。(17-9 节)

```
PS C:\Users\User> python d:\Python\ex\ex17_8.py
请扫描或输入车牌(Q/q代表结束) : atq9305.jpg
车辆入场时间 :  ATQ9305 : Mon Jun  3 15:06:43 2019
请扫描或输入车牌(Q/q代表结束) : rbt1388.jpg
车辆入场时间 :  RBT- 13838 : Mon Jun  3 15:06:56 2019
请扫描或输入车牌(Q/q代表结束) : atq9305.jpg
车辆出场时间 :  ATQ9305 : Mon Jun  3 15:07:09 2019
停车费用 :  60.0  元
请扫描或输入车牌(Q/q代表结束) : q
```

9. 请辨识下列繁体中文图片，然后存入 ex17_9.txt。(17-9 节)

2019 年 3 月
深智數位科技股份有限公司
台北市長安東路

```
PS C:\Users\User> python d:\Python\ex\ex17_9.py
2019 年 3 月
深 智 數 位 科 技 股 份 有 限 公 司
台 北 市 長 安 東 路
```

10. 请辨识下列简体中文图片，然后存入 ex17_10.txt。(17-9 节)

2019 年 3 月
深智数字科技股份有限公司
台北市长安东路

```
PS C:\Users\User> python d:\Python\ex\ex17_10.py
2019 年 3 月
深 智 数 字 科 技 股 份 有 限 公 司
台 北 市 长 安 东 路
```

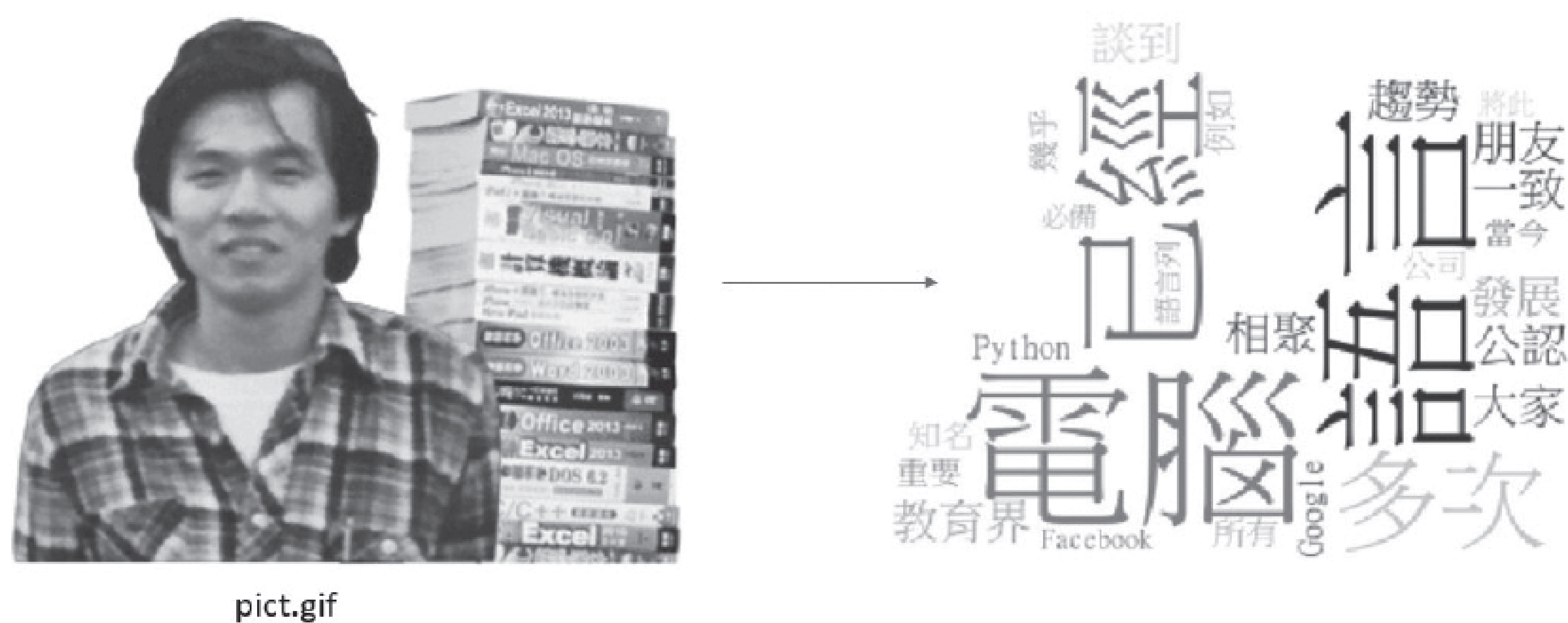
11. 请参考 ch17_28.py，建立含白色背景的词云。(17-10 节)



12. 请参考 ch17_28.py，建立含白色背景的词云，同时使用 OLDENGL.tff 文件，读者可以在 c:\Windows\Fonts 找到此文件。(17-10 节)

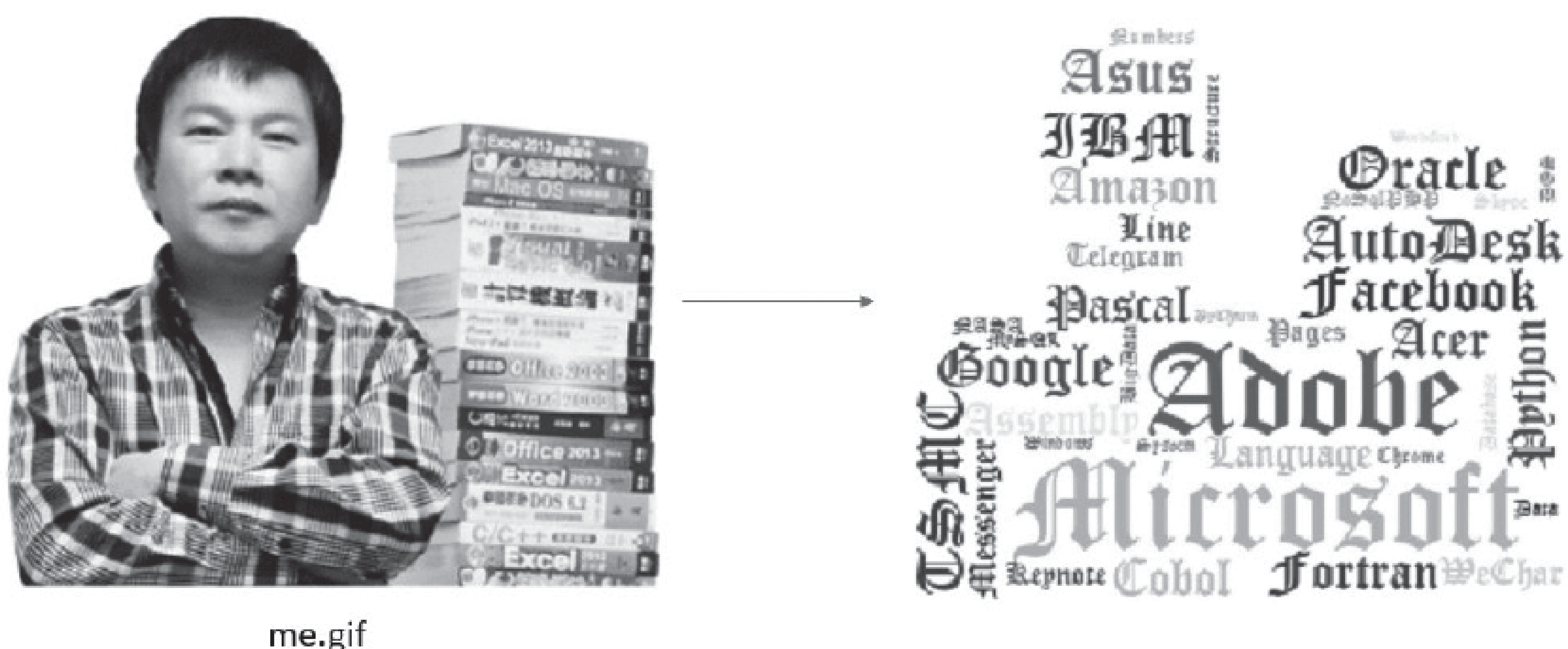


13. 请参考 ch17_34.py，然后建立词云图案，所使用的文件请自行设计，图片使用 pict.gif。(17-10 节)



pict.gif

14. 请参考 ch17_35.py，然后建立词云图案，所使用的文本文件请自行设计，此例笔者使用 text17_28.txt 文本文件，图片使用 me.gif。(17-10 节)



me.gif

18

第 18 章

使用 tkinter 开发 GUI 程序

本章摘要

GUI 英文全名是 Graphical User Interface，中文可以翻译为**图形用户接口**，本章将介绍使用 tkinter 模块设计这方面的程序。

Tk 是一个**开放源码**（open source）的图形接口的开发工具，最初发展是从 1991 年开始，具有跨平台的特性，可以在 Linux、Windows、Mac OS 等操作系统上执行。这个工具提供许多图形接口，例如，菜单（Menu）、按钮（Button）等。目前这个工具已经移植到 Python 语言，在 Python 语言中称为 tkinter 模块。

在安装 Python 时，就已经同时安装此模块了，在使用前只需声明导入此模块即可，如下所示。

```
from tkinter import *
```

注 在 Python 2 版本中模块名称是 Tkinter，Python 3 版本中的模块名称改为 tkinter。

18-1 建立窗口

可以使用下列方法建立窗口。

```
window = Tk( )                  # 这是自行定义的 Tk 对象名称，也可以取其他名称
window.mainloop( )              # 放在程序最后一行
```

通常我们将使用 Tk( ) 方法建立的窗口称为**根窗口**（root window），未来可以在此根窗口中建立许多**组件**（widget），甚至也可以在此根窗口建立上层窗口，此例笔者用 window 当作对象名称，读者也可以自行取其他名称。上述 mainloop( ) 方法可以让程序继续执行，同时进入等待与处理窗口事件，若是单击窗口右上方的“**关闭**”按钮，此程序才会结束。

程序实例 ch18_1.py：建立空白窗口。

```
1   # ch18_1.py
2   from tkinter import *
3
4   window = Tk()
5   window.mainloop()
```

执行结果



在上述窗口产生时，可以拖曳移动窗口或更改窗口大小，下列是与窗口相关的设置。

title( )：窗口标题。

geometry("widthxheight")：窗口的宽与高，单位是像素。

maxsize(width,height)：拖曳时可以设置窗口最大的宽（width）与高（height）。

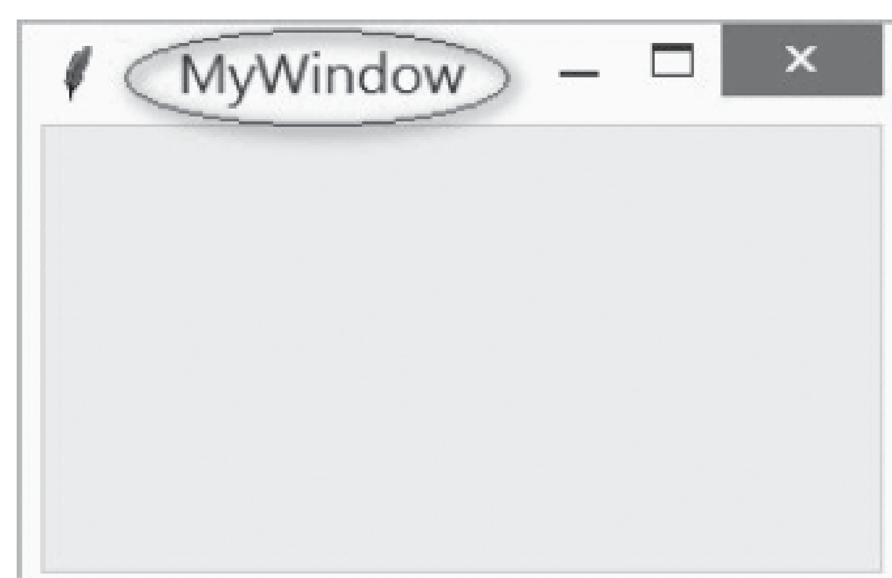
resizeable(True,True)：可设置是否更改窗口大小，第一个参数是宽，第二个参数是高，如果要

固定窗口宽与高，可以使用 resizeable(0,0)。

程序实例 ch18_2.py：建立窗口标题 MyWindow，同时设置宽是 300，高是 160。

```
1  # ch18_2.py
2  from tkinter import *
3
4  window = Tk()
5  window.title("MyWindow")      # 窗口标题
6  window.geometry("300x160")  # 窗口大小
7
8  window.mainloop()
```

执行结果



18-2 标签 Label

Label( ) 方法可以用于在窗口内建立**文字或图形**标签，有关图形标签将在 18-12 节讨论，它的使用格式如下。

Label(**父对象**，options, …)

Label( ) 方法的第一个参数是**父对象**，表示这个标签将建立在哪一个父对象（可想成父窗口或容器）内。下列是 Label( ) 方法内其他常用的 options 参数。

text：标签内容，如果有 "\n" 则可建立多行文字。

width：标签宽度，**单位是字符**。

height：标签高度，**单位是字符**。

bg 或 background：背景色彩。

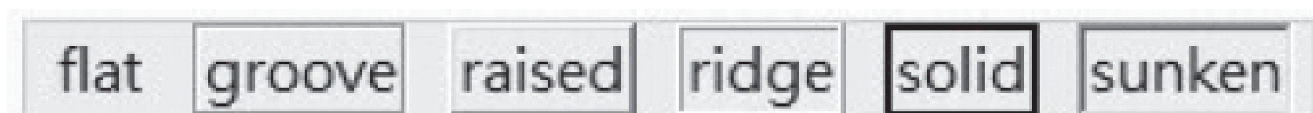
fg 或 foreground：字体色彩。

font( )：可选择字体与大小，可参考 ch18_4_1.py。

textvariable：可以设置标签以变量方式显示，可参考 ch18_14.py。

image：标签以图形方式呈现，将在 18-12 节解说。

relief：默认是 relief=flat，可由此控制标签的外框，有下列选项。

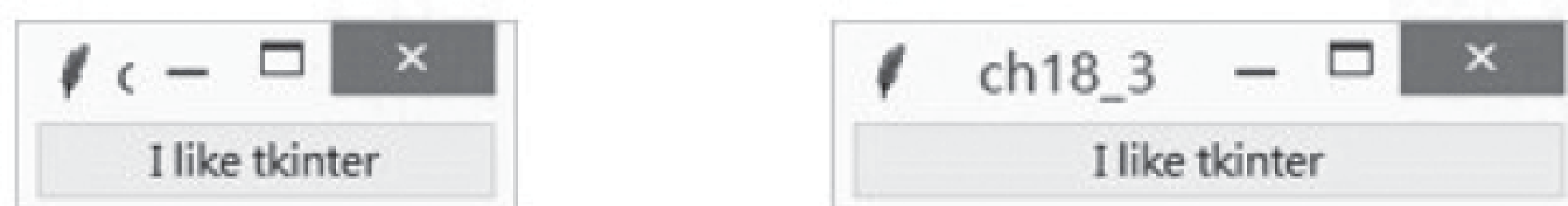


justify：在多行文件时最后一行的对齐方式有 LEFT/CENTER/RIGHT（靠左 / **居中** / 靠右），默认是**居中**对齐，将在 18-12-1 节以实例说明。

程序实例 ch18_3.py：建立一个标签，内容是 I like tkinter。

```
1  # ch18_3.py
2  from tkinter import *
3
4  window = Tk()
5  window.title("ch18_3")              # 窗口标题
6  label = Label(window,text="I like tkinter")
7  label.pack()                        # 包装与定位组件
8
9  window.mainloop()
```

执行结果 下方右图为鼠标拖曳增加窗口宽度的结果，可以看到完整的窗口标题。



上述第 7 行的 pack( ) 方法主要是**包装窗口的组件和定位窗口的对象**，所以可以在窗口内见到上述窗口组件，此例中窗口组件是标签。对上述第 6 行和第 7 行，也可以组合成一行，可参考下列程序实例。

程序实例 ch18_3_1.py：使用 Label( ).pack( ) 方式重新设计 ch18_3.py。

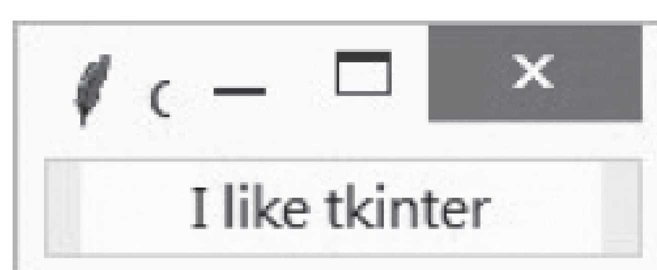
```
1  # ch18_3_1.py
2  from tkinter import *
3
4  window = Tk()
5  window.title("ch18_3_1")            # 窗口标题
6  label = Label(window,text="I like tkinter").pack()
7
8  window.mainloop()
```

执行结果 与 ch18_3.py 相同。

程序实例 ch18_4.py：扩充 ch18_3.py，标签宽度是 15，背景是浅黄色。

```
1  # ch18_4.py
2  from tkinter import *
3
4  window = Tk()
5  window.title("ch18_4")              # 窗口标题
6  label = Label(window,text="I like tkinter",
7                bg="lightyellow",     # 标签背景是浅黄色
8                width=15)             # 标签宽度是15
9  label.pack()                        # 包装与定位组件
10
11 window.mainloop()
```

执行结果



程序实例 ch18_4_1.py：重新设计 ch18_4.py，使用 font 更改字体与大小的应用。

```
1  # ch18_4_1.py
2  from tkinter import *
3
4  window = Tk()
5  window.title("ch18_4_1")                 # 窗口标题
6  label = Label(window,text="I like tkinter",
7                bg="lightyellow",          # 标签背景是浅黄色
8                width=15,                  # 标签宽度是15
9                font="Helvetica 16 bold italic")
10 label.pack()                             # 包装与定位组件
11
12 window.mainloop()
```

执行结果



上述语句最重要的是第 9 行，Helvetica 是字体名称，16 是字号，bold、italic 则是粗体与斜体，如果不设置则使用默认的一般字体。

18-3 窗口组件配置管理员

在设计 GUI 程序时，可以使用 3 种方法包装和定位各组件的位置，这 3 种方法又称为窗口组件配置管理员（Layout Management），下面将分成 3 节说明。

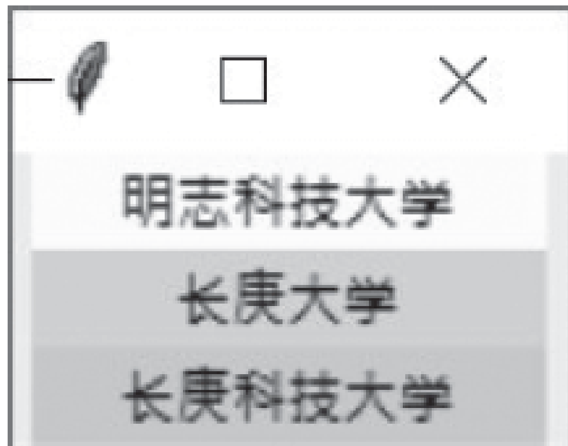
18-3-1 pack( ) 方法

在正式讲解 pack( ) 方法前，请先参考下列程序实例。

程序实例 ch18_5.py：一个窗口含 3 个标签的应用。

```
1  # ch18_5.py
2  from tkinter import *
3
4  window = Tk()
5  window.title("ch18_5")                   # 窗口标题
6  lab1 = Label(window,text="明志科技大学",
7               bg="lightyellow",           # 标签背景是浅黄色
8               width=15)                   # 标签宽度是15
9  lab2 = Label(window,text="长庚大学",
10              bg="lightgreen",            # 标签背景是浅绿色
11              width=15)                   # 标签宽度是15
12 lab3 = Label(window,text="长庚科技大学",
13              bg="lightblue",             # 标签背景是浅蓝色
14              width=15)                   # 标签宽度是15
15 lab1.pack()                              # 包装与定位组件
16 lab2.pack()                              # 包装与定位组件
17 lab3.pack()                              # 包装与定位组件
18
19 window.mainloop()
```

执行结果



由上图可以看到当窗口有多个组件时，使用 pack( ) 可以让组件由上往下排列然后显示，其实这也是系统的默认环境。使用 pack( ) 方法时，也可以增加 side 参数设置组件的排列方式，此参数的值如下。

TOP：这是默认值，由上往下排列。

BOTTOM：由下往上排列。

LEFT：由左往右排列。

RIGHT：由右往左排列。

另外，使用 pack( ) 方法时，窗口组件间的距离是 1 像素，如果需要有适度间距，可以增加参数 padx/pady，代表水平间距 / 垂直间距，可以分别在组件间增加间距。

程序实例 ch18_6.py：在 pack( ) 方法内增加 side=BOTTOM 重新设计 ch18_5.py。

```
15  lab1.pack(side=BOTTOM)                # 包装与定位组件
16  lab2.pack(side=BOTTOM)                # 包装与定位组件
17  lab3.pack(side=BOTTOM)                # 包装与定位组件
```

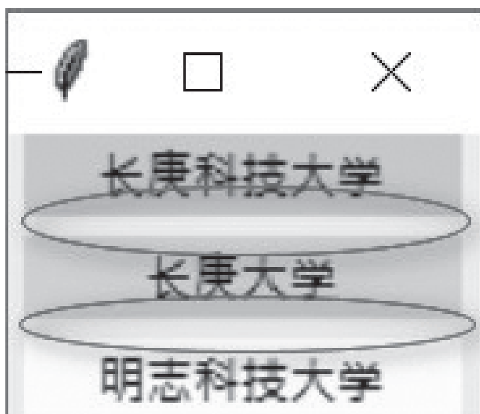
执行结果



程序实例 ch18_6_1.py：重新设计 ch18_6.py，在“长庚大学”标签上下增加 5 像素间距。

```
15  lab1.pack(side=BOTTOM)                # 包装与定位组件
16  lab2.pack(side=BOTTOM,pady=5)         # 包装与定位组件,增加y轴间距
17  lab3.pack(side=BOTTOM)                # 包装与定位组件
```

执行结果



程序实例 ch18_7.py：在 pack( ) 方法内增加 side=LEFT 重新设计 ch18_5.py。

```
15  lab1.pack(side=LEFT)                  # 包装与定位组件
16  lab2.pack(side=LEFT)                  # 包装与定位组件
17  lab3.pack(side=LEFT)                  # 包装与定位组件
```

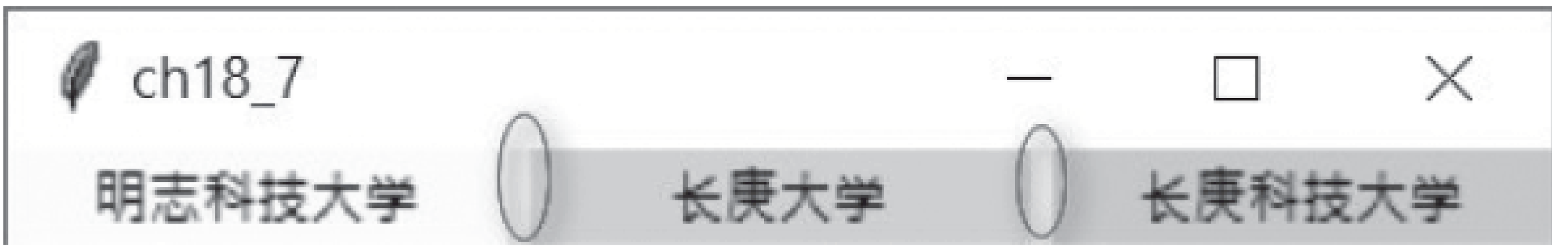
执行结果



程序实例 ch18_7_1.py：重新设计 ch18_5.py，在"长庚大学"标签左右增加 5 像素间距。

```
15  lab1.pack(side=LEFT)                  # 包装与定位组件
16  lab2.pack(side=LEFT,padx=5)           # 包装与定位组件,增加x轴间距
17  lab3.pack(side=LEFT)                  # 包装与定位组件
```

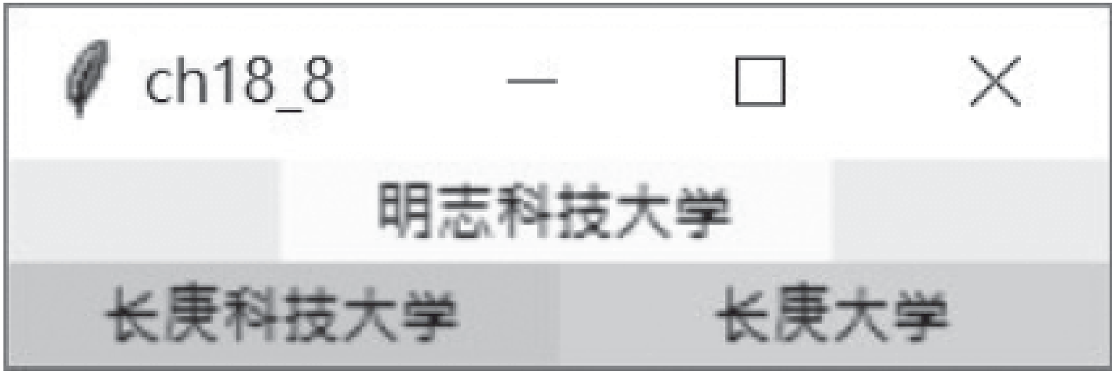
执行结果



程序实例 ch18_8.py：在 pack( ) 方法内混合使用 side 参数重新设计 ch18_5.py。

```
15  lab1.pack()                          # 包装与定位组件
16  lab2.pack(side=RIGHT)                # 包装与定位组件
17  lab3.pack(side=LEFT)                 # 包装与定位组件
```

执行结果



18-3-2　grid( ) 方法

1. 基本概念

这是一种以格状（类似 Excel 电子表格）包装和定位窗口组件的方法，使用 row 和 column 参数。下面是此格状方法的概念。

row=0,column=0	row=0,column=1	..	row=0,column=n
row=1,column=0	row=1,column=1	..	row=1,column=n
:	:		:
row=n,column=0	row=n,column=1	..	row=n,column=n

注　上述表达方式也可以将最左上角的 row 和 column 从 1 开始计数。

可以适度调整 grid( ) 方法内的 row 和 column 值，即可包装窗口组件的位置。

程序实例 ch18_9.py：使用 grid( ) 方法取代 pack( ) 方法重新设计 ch18_5.py。

```
15  lab1.grid(row=0,column=0)           # 格状包装
16  lab2.grid(row=1,column=0)           # 格状包装
17  lab3.grid(row=1,column=1)           # 格状包装
```

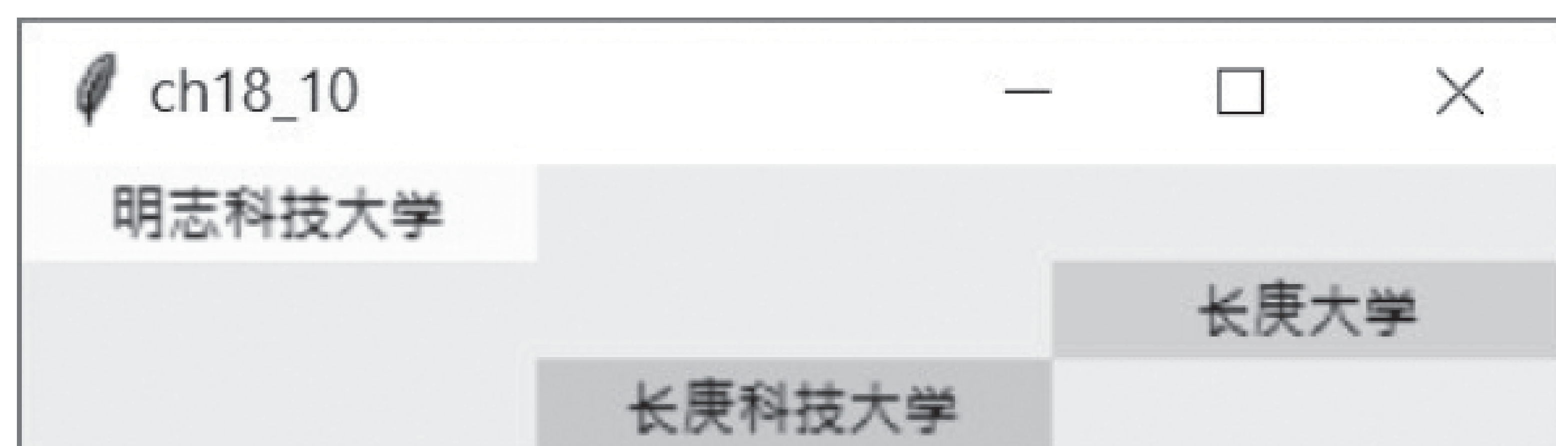
执行结果



程序实例 ch18_10.py：格状包装的另一个应用。

```
15  lab1.grid(row=0,column=0)           # 格状包装
16  lab2.grid(row=1,column=2)           # 格状包装
17  lab3.grid(row=2,column=1)           # 格状包装
```

执行结果



在 grid( ) 方法内也可以增加 sticky 参数，可以用此参数设置 N/S/W/E，意义是上 / 下 / 左 / 右对齐。此外，也可以增加 padx/pady 参数分别设置组件与相邻组件的 x 轴间距 /y 轴间距。细节可以参考程序实例 ch18_17.py。

2. columnspan 参数

可以设置控件在 column 方向的合并数量，在正式讲解 columnspan 参数功能前，下面先介绍建立一个含 8 个标签的应用。

程序实例 ch18_10_1.py：使用 grid 方法建立含 8 个标签的应用。

```
1   # ch18_10_1.py
2   from tkinter import *
3
4   window = Tk()
5   window.title("ch18_10_1")                  # 窗口标题
6   lab1 = Label(window,text="标签1",relief="raised")
7   lab2 = Label(window,text="标签2",relief="raised")
8   lab3 = Label(window,text="标签3",relief="raised")
9   lab4 = Label(window,text="标签4",relief="raised")
10  lab5 = Label(window,text="标签5",relief="raised")
11  lab6 = Label(window,text="标签6",relief="raised")
12  lab7 = Label(window,text="标签7",relief="raised")
13  lab8 = Label(window,text="标签8",relief="raised")
14  lab1.grid(row=0,column=0)
15  lab2.grid(row=0,column=1)
16  lab3.grid(row=0,column=2)
17  lab4.grid(row=0,column=3)
18  lab5.grid(row=1,column=0)
19  lab6.grid(row=1,column=1)
20  lab7.grid(row=1,column=2)
21  lab8.grid(row=1,column=3)
22
23  window.mainloop()
```

执行结果



如果发生了标签 2 和标签 3 的区间是被一个标签占用的情况，此时就是使用 columnspan 参数的场合。

程序实例 ch18_10_2.py：重新设计 ch18_10_1.py，将标签 2 和标签 3 合并成一个标签。

```
1  # ch18_10_2.py
2  from tkinter import *
3
4  window = Tk()
5  window.title("ch18_10_2")              # 窗口标题
6  lab1 = Label(window,text="标签1",relief="raised")
7  lab2 = Label(window,text="标签2",relief="raised")
8  lab4 = Label(window,text="标签4",relief="raised")
9  lab5 = Label(window,text="标签5",relief="raised")
10 lab6 = Label(window,text="标签6",relief="raised")
11 lab7 = Label(window,text="标签7",relief="raised")
12 lab8 = Label(window,text="标签8",relief="raised")
13 lab1.grid(row=0,column=0)
14 lab2.grid(row=0,column=1,columnspan=2)
15 lab4.grid(row=0,column=3)
16 lab5.grid(row=1,column=0)
17 lab6.grid(row=1,column=1)
18 lab7.grid(row=1,column=2)
19 lab8.grid(row=1,column=3)
20
21 window.mainloop()
```

执行结果



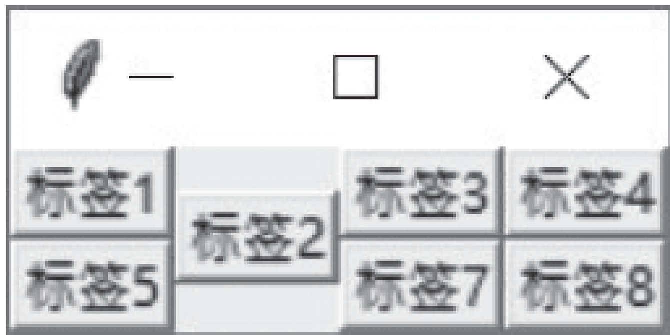
3. rowspan 参数

可以设置控件在 row 方向的合并数量。参照程序实例 ch18_10_1.py，如果发生了标签 2 和标签 6 的区间是被同一个标签占用，此时就是使用 rowspan 参数的场合。

程序实例 ch18_10_3.py：重新设计 ch18_10_1.py，将标签 2 和标签 6 合并成一个标签。

```
1  # ch18_10_3.py
2  from tkinter import *
3
4  window = Tk()
5  window.title("ch18_10_3")              # 窗口标题
6  lab1 = Label(window,text="标签1",relief="raised")
7  lab2 = Label(window,text="标签2",relief="raised")
8  lab3 = Label(window,text="标签3",relief="raised")
9  lab4 = Label(window,text="标签4",relief="raised")
10 lab5 = Label(window,text="标签5",relief="raised")
11 lab7 = Label(window,text="标签7",relief="raised")
12 lab8 = Label(window,text="标签8",relief="raised")
13 lab1.grid(row=0,column=0)
14 lab2.grid(row=0,column=1,rowspan=2)
15 lab3.grid(row=0,column=2)
16 lab4.grid(row=0,column=3)
17 lab5.grid(row=1,column=0)
18 lab7.grid(row=1,column=2)
19 lab8.grid(row=1,column=3)
20
21 window.mainloop()
```

执行结果



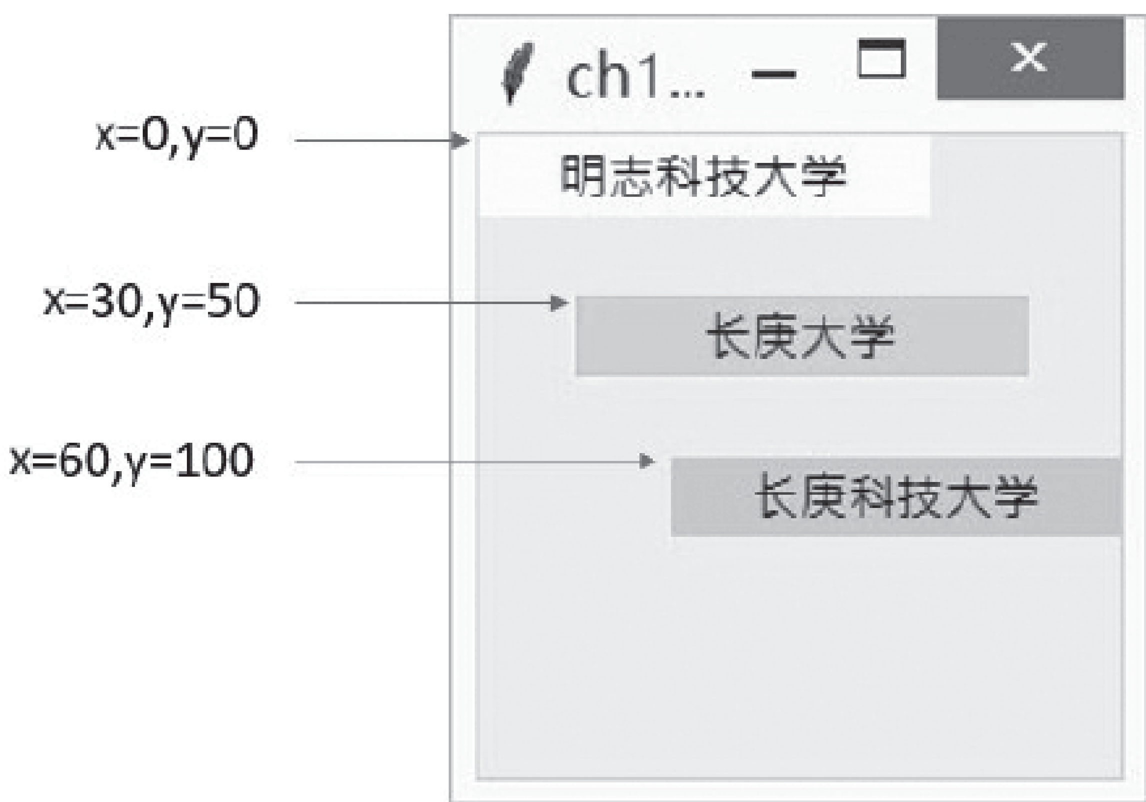
18-3-3 place( ) 方法

使用 place( ) 方法内的 x 和 y 参数可以直接设置窗口组件的**左上方位置**，单位是**像素**，窗口显示区的左上角是（x=0,y=0），x 是往右递增，y 是往下递增。使用这种方法时，窗口将不会自动重设大小，而是使用默认的大小显示，可参考 ch18_1.py 的执行结果。

程序实例 ch18_11.py：使用 place( ) 方法直接设置标签的位置，重新设计 ch18_5.py。

```
 1  # ch18_11.py
 2  from tkinter import *
 3
 4  window = Tk()
 5  window.title("ch18_11")                  # 窗口标题
 6  lab1 = Label(window,text="明志科技大学",
 7               bg="lightyellow",           # 标签背景是浅黄色
 8               width=15)                   # 标签宽度是15
 9  lab2 = Label(window,text="长庚大学",
10               bg="lightgreen",            # 标签背景是浅绿色
11               width=15)                   # 标签宽度是15
12  lab3 = Label(window,text="长庚科技大学",
13               bg="lightblue",             # 标签背景是浅蓝色
14               width=15)                   # 标签宽度是15
15  lab1.place(x=0,y=0)                      # 直接定位
16  lab2.place(x=30,y=50)                    # 直接定位
17  lab3.place(x=60,y=100)                   # 直接定位
18
19  window.mainloop()
```

执行结果



18-3-4 窗口组件位置的总结

使用 tkinter 模块设计 GUI 程序时，虽然可以使用 place( ) 方法定位组件的位置，不过笔者建议尽量使用 pack( ) 和 grid( ) 方法定位组件的位置，因为当窗口组件较多时，使用 place( ) 需计算组件位置，会比较不方便，同时当新增或减少组件时又需重新计算设置组件位置，这样较为不便。

18-4 功能按钮 Button

18-4-1 基本概念

功能按钮也可称为按钮，在窗口组件中可以设计单击功能按钮时，执行某一个特定的动作。它的使用格式如下。

Button (**父对象** , options, …)

Button( ) 方法的第一个参数是**父对象**，表示这个功能按钮将建立在哪一个窗口内。下列是 Button( ) 方法内其他常用的 options 参数。

text：功能按钮名称。

width：宽，单位是字符宽。

height：高，单位是字符高。

bg 或 background：背景色彩。

fg 或 foreground：字体色彩。

image：功能按钮上的图形，可参考 18-12-2 节。

command：单击功能按钮时，执行此参数所指定的方法。

程序实例 ch18_12.py：当单击功能按钮时可以显示字符串 I love Python，底色是浅黄色，字符串颜色是蓝色。

```
1  # ch18_12.py
2  from tkinter import *
3
4  def msgShow():
5      label["text"] = "I love Python"
6      label["bg"] = "lightyellow"
7      label["fg"] = "blue"
8
9  window = Tk()
10 window.title("ch18_12")             # 窗口标题
11 label = Label(window)               # 标签内容
12 btn = Button(window,text="Message",command=msgShow)
13
14 label.pack()
15 btn.pack()
16
17 window.mainloop()
```

执行结果



程序实例 ch18_13.py：扩充设计 ch18.12.py，若单击 Exit 按钮，窗口可以关闭。

```
1  # ch18_13.py
2  from tkinter import *
3
4  def msgShow():
5      label["text"] = "I love Python"
6      label["bg"] = "lightyellow"
7      label["fg"] = "blue"
8
9  window = Tk()
10 window.title("ch18_13")              # 窗口标题
11 label = Label(window)                # 标签内容
12 btn1 = Button(window,text="Message",width=15,command=msgShow)
13 btn2 = Button(window,text="Exit",width=15,command=window.destroy)
14 label.pack()
15 btn1.pack(side=LEFT)                 # 按钮1
16 btn2.pack(side=RIGHT)                # 按钮2
17
18 window.mainloop()
```

执行结果



上述第 13 行的 window.destroy 可以关闭 window 窗口对象，同时程序结束。另一个常用的是 window.quit，可以让 Python Shell 内执行的程序结束，但是 window 窗口则继续执行，在 ch18_16.py 中会做实例说明。

18-4-2 设置窗口背景 config( )

config（option=value）其实是窗口组件的通用方法，通过设置 option 为 bg 参数时，可以设置窗口组件的背景颜色。

程序实例 ch18_13_1.py：在窗口右下角有 3 个按钮，单击 Yellow 按钮可以将窗口背景设为黄色，单击 Blue 按钮可以将窗口背景设为蓝色，单击 Exit 按钮可以结束程序。

```
1  # ch18_13_1.py
2  from tkinter import *
3
4  def yellow():                        # 设置窗口背景是黄色
5      window.config(bg="yellow")
6  def blue():                          # 设置窗口背景是蓝色
7      window.config(bg="blue")
8
9  window = Tk()
10 window.title("ch18_13_1")
11 window.geometry("300x200")            # 固定窗口大小
12 # 依次建立3个按钮
13 exitbtn = Button(window,text="Exit",command=window.destroy)
14 bluebtn = Button(window,text="Blue",command=blue)
15 yellowbtn = Button(window,text="Yellow",command=yellow)
16 # 将3个按钮包装定位在右下方
17 exitbtn.pack(anchor=S,side=RIGHT,padx=5,pady=5)
18 bluebtn.pack(anchor=S,side=RIGHT,padx=5,pady=5)
19 yellowbtn.pack(anchor=S,side=RIGHT,padx=5,pady=5)
20
21 window.mainloop()
```

执行结果



18-4-3　使用 lambda 表达式的好时机

在 ch18_13_1.py 设计过程中，Yellow 按钮和 Blue 按钮是执行相同工作，但是所传递的颜色参数不同，其实这是使用 lambda 表达式的好时机，我们可以通过 lambda 表达式调用相同的方法，但是传递不同的参数方式简化设计。

程序实例 ch18_13_2.py：使用 lambda 表达式重新设计 ch18_13_1.py。

```
1  # ch18_13_2.py
2  from tkinter import *
3
4  def bColor(bgColor):                # 设置窗口背景颜色
5      window.config(bg=bgColor)
6
7  window = Tk()
8  window.title("ch18_13_2")
9  window.geometry("300x200")          # 固定窗口大小
10 # 依次建立3个按钮
11 exitbtn = Button(window,text="Exit",command=window.destroy)
12 bluebtn = Button(window,text="Blue",command=lambda:bColor("blue"))
13 yellowbtn = Button(window,text="Yellow",command=lambda:bColor("yellow"))
14 # 将3个按钮包装定位在右下方
15 exitbtn.pack(anchor=S,side=RIGHT,padx=5,pady=5)
16 bluebtn.pack(anchor=S,side=RIGHT,padx=5,pady=5)
17 yellowbtn.pack(anchor=S,side=RIGHT,padx=5,pady=5)
18
19 window.mainloop()
```

上述也可以省略第 4、5 行的 bColor( ) 函数，此时第 12、13 行的 lambda 将改成下列语句。

```
command=lambda:window.config(bg="blue")
command=lambda:window.config(bg="yellow")
```

18-5　变量类型

有些窗口组件在执行时会更改内容，此时可以使用 tkinter 模块内的变量类型（Variable Classes），它的使用方式如下。

```
x = IntVar( )              # 整数变量，默认是 0
x = DoubleVar( )           # 浮点数变量，默认是 0.0
x = StringVar( )           # 字符串变量，默认是 ""
```

x = BooleanVar( )　　　　　　　# 布尔值变量，True 是 1，False 是 0

可以使用 get( ) 方法取得变量内容，使用 set( ) 方法设置变量内容。

程序实例 ch18_14.py：这个程序在执行时若单击 Hit 按钮可以显示 I like tkinter 字符串，如果已经显示此字符串则改成不显示此字符串。这个程序第 17 行是将标签内容设为变量 x，第 8 行是设置显示标签时的标签内容，第 11 行则是将标签内容设为空字符串，如此可以达到不显示标签内容。

```
1  # ch18_14.py
2  from tkinter import *
3
4  def btn_hit():                          # 处理按钮事件
5      global msg_on                       # 这是全局变量
6      if msg_on == False:
7          msg_on = True
8          x.set("I like tkinter")         # 显示文字
9      else:
10         msg_on = False
11         x.set("")                       # 不显示文字
12
13 window = Tk()
14 window.title("ch18_14")                 # 窗口标题
15
16 msg_on = False                          # 全局变量默认是False
17 x = StringVar()                         # Label的变量内容
18
19 label = Label(window,textvariable=x,    # 设置Label内容是变量x
20               fg="blue",bg="lightyellow", # 浅黄色底蓝色字
21               font="Verdana 16 bold",   # 字体设置
22               width=25,height=2).pack() # 标签内容
23 btn = Button(window,text="Hit",command=btn_hit).pack()
24
25 window.mainloop()
```

执行结果





18-6 文本框 Entry

文本框 Entry 通常是指单行的文本框，它的使用格式如下。

Entry(父对象 , options, …)

Entry( ) 方法的第一个参数是父对象，表示这个文本框将建立在哪一个窗口内。下列是 Entry( ) 方法内其他常用的 options 参数。

width：宽，单位是字符宽。

height：高，单位是字符高。

bg 或 background：背景色彩。

fg 或 foreground：字体色彩。

state：输入状态，默认是 NORMAL，表示可以输入，DISABLE 则是无法输入。

textvariable：文字变量。

show：显示输入字符，例如，show='*' 表示显示星号，常用于密码字段输入。

程序实例 ch18_15.py：在窗口内建立标签和文本框，读者也可以在文本框内执行输入，其中第 2 个文本框对象 e2 有设置 show='*'，所以输入时所输入的字符用 * 显示。

```
1   # ch18_15.py
2   from tkinter import *
3
4   window = Tk()
5   window.title("ch18_15")          # 窗口标题
6
7   lab1 = Label(window,text="Account ").grid(row=0)
8   lab2 = Label(window,text="Password").grid(row=1)
9
10  e1 = Entry(window)               # 文本框1
11  e2 = Entry(window,show='*')      # 文本框2
12  e1.grid(row=0,column=1)          # 定位文本框1
13  e2.grid(row=1,column=1)          # 定位文本框2
14
15  window.mainloop()
```

执行结果

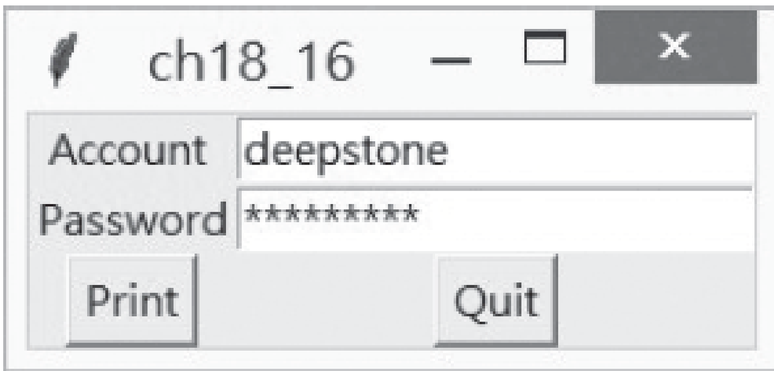


上述第 7 行设置 grid(row=0)，在没有设置 column=x 的情况下，系统将自动设置 column=0，第 8 行相同。

程序实例 ch18_16.py：扩充上述程序，增加 Print 按钮和 Quit 单击，若是单击 Print 按钮，可以在 Python Shell 窗口看到所输入的 Account 和 Password。若是单击 Quit 按钮，可以看到在 Python Shell 窗口执行的程序结束，但是屏幕上仍可以看到此 ch18_16 窗口在执行。

```
1   # ch18_16.py
2   from tkinter import *
3   def printInfo():                       # 打印输入信息
4       print("Account: %s\nPassword: %s" % (e1.get(),e2.get()))
5
6   window = Tk()
7   window.title("ch18_16")                # 窗口标题
8
9   lab1 = Label(window,text="Account ").grid(row=0)
10  lab2 = Label(window,text="Password").grid(row=1)
11
12  e1 = Entry(window)                     # 文本框1
13  e2 = Entry(window,show='*')            # 文本框2
14  e1.grid(row=0,column=1)                # 定位文本框1
15  e2.grid(row=1,column=1)                # 定位文本框2
16
17  btn1 = Button(window,text="Print",command=printInfo)
18  btn1.grid(row=2,column=0)
19  btn2 = Button(window,text="Quit",command=window.quit)
20  btn2.grid(row=2,column=1)
21
22  window.mainloop()
```

执行结果



下面是先单击 Print 按钮，再单击 Quit 按钮，在 Python Shell 窗口的执行结果。

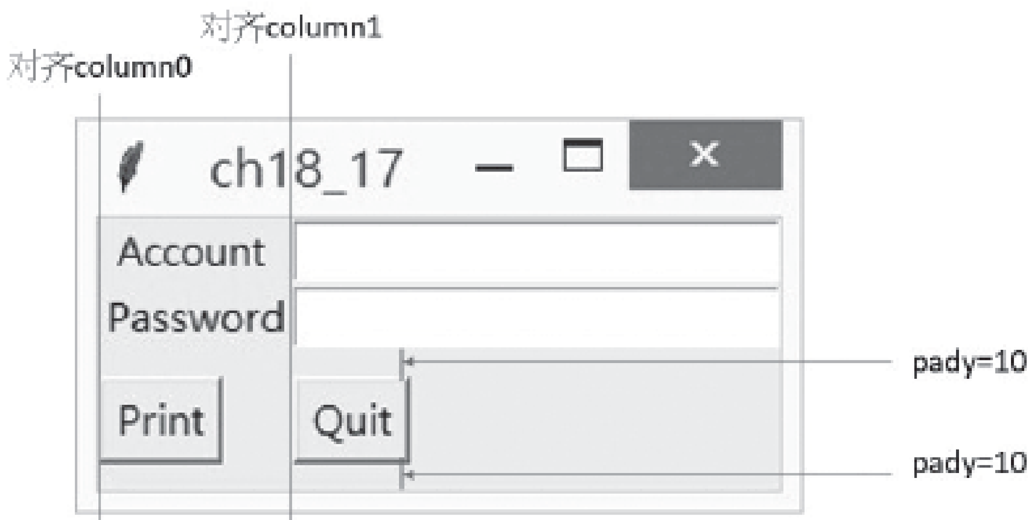
```
==================== RESTART: D:\Python\ch18\ch18_16.py ====================
Account: deepstone
Password: deepstone
>>>
```

从上述执行结果可以看到，Print 按钮和 Quit 按钮并没有对齐上方的标签和文本框，我们可以在 grid( ) 方法内增加 sticky 参数，同时将此参数设为 W，即可靠左对齐字段。另外，也可以使用 pady 设置对象上下的间距，padx 则是可以设置左右的间距。

程序实例 ch18_17.py：使用 sticky=W 参数和 pady=10 参数，重新设计 ch18_16.py。

```
17  btn1 = Button(window,text="Print",command=printInfo)
18  # sticky=W可以设置对象与上面的Label对齐, pady设置上下间距是10
19  btn1.grid(row=2,column=0,sticky=W,pady=10)
20  btn2 = Button(window,text="Quit",command=window.quit)
21  # sticky=W可以设置对象与上面的Entry对齐, pady设置上下间距是10
22  btn2.grid(row=2,column=1,sticky=W,pady=10)
```

执行结果



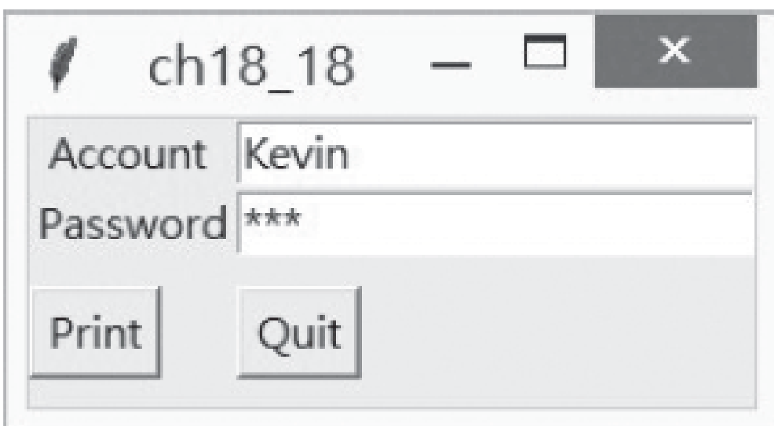
1. 在 Entry 中插入字符串

在 tkinter 模块的应用中可以使用 insert(index,s) 方法插入字符串，s 是所插入的字符串，字符串会插入在 index 位置前。程序设计时可以使用这个方法为文本框建立默认的文字，通常会将它放在 Entry( ) 方法建立完文本框后，可参考下列实例第 14、15 行。

程序实例 ch18_18.py：扩充 ch18_17.py，为程序建立默认的 Account 为 kevin，Password 为 pwd。相较于 ch18_17.py 这个程序增加了第 14、15 行。

```
12  e1 = Entry(window)                  # 文本框1
13  e2 = Entry(window,show='*')         # 文本框2
14  e1.insert(1,"Kevin")                # 默认文本框1内容
15  e2.insert(1,"pwd")                  # 默认文本框2内容
```

执行结果



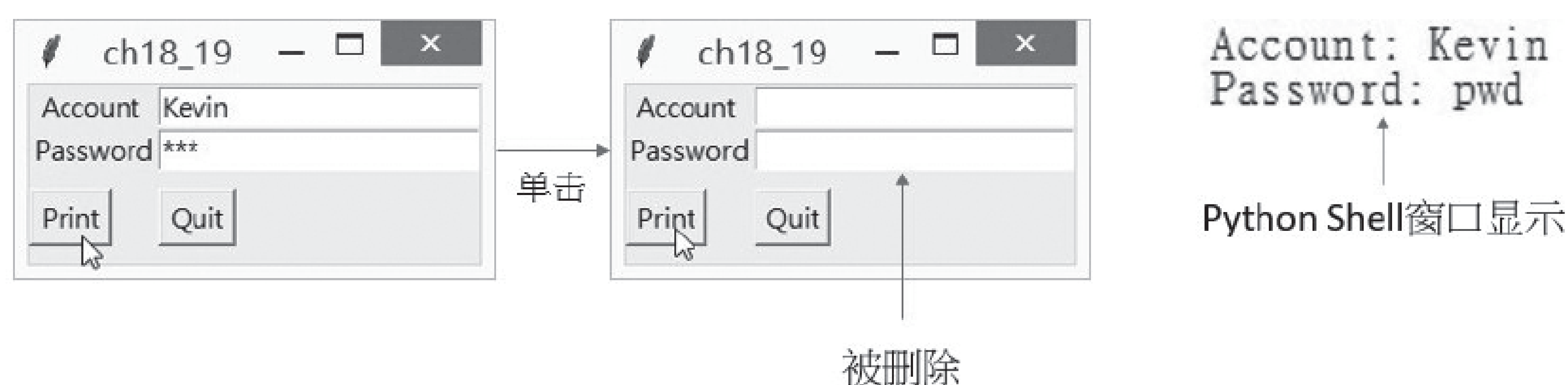
2. 在 Entry 中删除字符串

在 tkinter 模块的应用中可以使用 delete(first,last=None) 方法删除 Entry 内的字符串，如果要删除整个字符串可以使用 delete(0,END)。

程序实例 ch18_19.py：扩充程序实例 ch18_18.py，当单击 Print 按钮后，清空文本框 Entry 中的内容。

```
 1  # ch18_19.py
 2  from tkinter import *
 3  def printInfo():                         # 打印输入信息
 4      print("Account: %s\nPassword: %s" % (e1.get(),e2.get()))
 5      e1.delete(0,END)                     # 删除文本框1
 6      e2.delete(0,END)                     # 删除文本框2
 7
 8  window = Tk()
 9  window.title("ch18_19")                  # 窗口标题
10
11  lab1 = Label(window,text="Account ").grid(row=0)
12  lab2 = Label(window,text="Password").grid(row=1)
13
14  e1 = Entry(window)                       # 文本框1
15  e2 = Entry(window,show='*')              # 文本框2
16  e1.insert(1,"Kevin")                     # 默认文本框1内容
17  e2.insert(1,"pwd")                       # 默认文本框2内容
18  e1.grid(row=0,column=1)                  # 定位文本框1
19  e2.grid(row=1,column=1)                  # 定位文本框2
20
21  btn1 = Button(window,text="Print",command=printInfo)
22  # sticky=W可以设置对象与上面的Label对齐, pady设置上下间距是10
23  btn1.grid(row=2,column=0,sticky=W,pady=10)
24  btn2 = Button(window,text="Quit",command=window.quit)
25  # sticky=W可以设置对象与上面的Entry对齐, pady设置上下间距是10
26  btn2.grid(row=2,column=1,sticky=W,pady=10)
27
28  window.mainloop()
```

执行结果



3. Entry 的应用

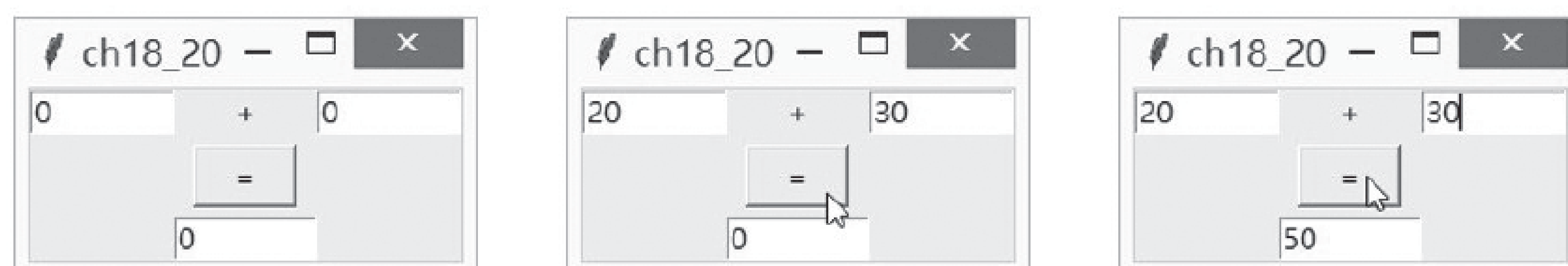
在结束本节前，将讲解标签、文本框、按钮的综合应用，当读者彻底了解了本程序后，就应该有能力设计小计算器程序了。

程序实例 ch18_20.py：设计可以执行加法运算的程序。

```
 1  # ch18_20.py
 2  from tkinter import *
 3  def add():                                   # 加法运算
 4      n3.set(n1.get()+n2.get())
 5
 6  window = Tk()
 7  window.title("ch18_20")                      # 窗口标题
 8
 9  n1 = IntVar()
10  n2 = IntVar()
11  n3 = IntVar()
12
```

```
13  e1 = Entry(window,width=8,textvariable=n1)        # 文本框1
14  label = Label(window,width=3,text='+')            # 加号
15  e2 = Entry(window,width=8,textvariable=n2)        # 文本框2
16  btn = Button(window,width=5,text='=',command=add) # =按钮
17  e3 = Entry(window,width=8,textvariable=n3)        # 存储结果文本框
18
19  e1.grid(row=0,column=0)                           # 定位文本框1
20  label.grid(row=0,column=1,padx=5)                 # 定位加号
21  e2.grid(row=0,column=2)                           # 定位文本框2
22  btn.grid(row=1,column=1,pady=5)                   # 定位=按钮
23  e3.grid(row=2,column=1)                           # 定位存储结果
24
25  window.mainloop()
```

执行结果 下列分别是程序执行前、输入数值、单击等号按钮的结果。



上述语句第 20 行内有 padx=5，相当于设置加号标签左右间距是 5 像素。第 22 行的 pady=5 是设置等号按钮上下间距是 5 像素。当我们单击等号按钮时，程序会执行第 3 行的 add( ) 函数执行加法运算，在此函数的 n1.get( ) 可以取得 n1 变量值，n3.set( ) 则是设置 n3 变量值。

18-7 文字区域 Text

可以将 Text 想成是 Entry 的扩充，可以在此输入多行文本，甚至也可以使用此区域建立简单的文字编辑程序或是利用它设计网页浏览程序。它的使用格式如下。

Text (**父对象** , options, …)

Text( ) 方法的第一个参数是**父对象**，表示这个文字区域将建立在哪一个窗口内。下列是 Text( ) 方法内其他常用的 options 参数。

width：宽，单位是字符宽。

height：高，单位是字符高。

bg 或 background：背景色彩。

fg 或 foreground：字体色彩。

state：输入状态，默认是 NORMAL，表示可以输入，DISABLE 则是无法输入。

xscrollbarcommand：水平滚动条。

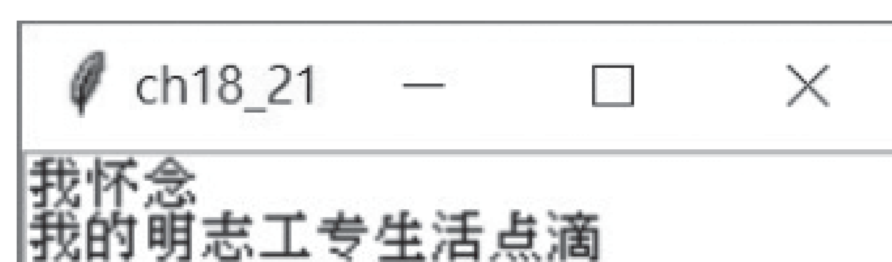
yscrollbarcommand：垂直滚动条，可参考下一节的实例。

wrap：这是换行参数，默认是 CHAR，如果输入数据超出行宽度时，必要时会将单字依拼音拆成不同行输出。如果是 WORD，则不会将单字拆成不同行输出。如果是 NONE，则不换行，这时将有水平滚动条。

程序实例 ch18_21.py：文字区域 Text 的基本应用。

```
1  # ch18_21.py
2  from tkinter import *
3
4  window = Tk()
5  window.title("ch18_21")                # 窗口标题
6
7  text = Text(window,height=2,width=30)
8  text.insert(END,"我怀念\n我的明志工专生活点滴")
9  text.pack()
10
11 window.mainloop()
```

执行结果



上述 insert( ) 方法的第一个参数 END 表示插入文字区域末端，由于目前文字区域是空的，所以就插在前面。

程序实例 ch18_22.py：插入多个字符串，发现文字区域不够使用，造成部分字符串无法显示。

```
1  # ch18_22.py
2  from tkinter import *
3
4  window = Tk()
5  window.title("ch18_22")                # 窗口标题
6
7  text = Text(window,height=2,width=30)
8  text.insert(END,"我怀念\n一个人的极境旅行")
9  str = """2016年12月,我一个人订了机票和船票,
10 开始我的南极旅行,飞机经迪拜再往阿根廷的乌斯怀雅,
11 在此我登上邮轮开始我的南极之旅"""
12 text.insert(END,str)
13 text.pack()
14
15 window.mainloop()
```

执行结果



由上述执行结果可以发现，字符串 str 中许多内容没有显示，此时可以增加文字区域 Text 的行数；另一种方法是可以使用滚动条，其实这也是比较高明的方法。

18-8 滚动条 Scrollbar

对前一节的实例而言，窗口内只有文字区域 Text，所以在设计滚动条时，可以只有一个参数，就是窗口对象，前面实例中均使用 window 当作窗口对象，此时可以用下列指令设计滚动条。

```
scrollbar = Scrollbar(window)            # scrollbar 是滚动条对象
```

程序实例 ch18_23.py：扩充程序实例 ch18_22.py，主要是增加滚动条功能。

```
1  # ch18_23.py
2  from tkinter import *
3
4  window = Tk()
5  window.title("ch18_23")                    # 窗口标题
6  scrollbar = Scrollbar(window)              # 卷轴对象
7  text = Text(window,height=2,width=30)      # 文字区域对象
8  scrollbar.pack(side=RIGHT,fill=Y)          # 靠右安置与父对象高度相同
9  text.pack(side=LEFT,fill=Y)                # 靠左安置与父对象高度相同
10 scrollbar.config(command=text.yview)
11 text.config(yscrollcommand=scrollbar.set)
12 text.insert(END,"我怀念\n一个人的极境旅行")
13 str = """2016年12月,我一个人订了机票和船票,
14 开始我的南极旅行,飞机经迪拜再往阿根廷的乌斯怀雅,
15 在此我登上邮轮开始我的南极之旅"""
16 text.insert(END,str)
17
18 window.mainloop()
```

执行结果



上述程序第 8 和 9 行的 fill=Y 主要是设置此对象高度与父对象相同，第 10 行 scrollbar.config( ) 方法主要是为 scrollbar 对象设置选择性参数内容，此例是设置 command 参数，它的用法与下列语句相同。

```
scrollbar["command"] = text.yview          # 设置执行方法
```

也就是当移动滚动条时，会去执行所指定的方法，此例是执行 yview( ) 方法。第 11 行是将文字区域的选项参数 yscrollcommand 设置为 scrollbar.set，表示将文字区域与滚动条做链接。

18-9 选项按钮 Radiobutton

选项按钮 Radio Button 名称的由来是无线电的按钮，在收音机时代可以用无线电的按钮选择特定频道。选项按钮最大的特点可以用鼠标单击方式选取此选项，同时一次只能有一个选项被选取。例如，在填写学历栏时，如果有一系列选项是高中、大学、硕士、博士，此时只能勾选一个项目。可以使用 Radiobutton( ) 方法建立选项按钮，它的使用方法如下。

Radiobutton(**父对象**, options, …)

Radiobutton( ) 方法的第一个参数是**父对象**，表示这个选项按钮将建立在哪一个窗口内。下列是 Radiobutton( ) 方法内其他常用的 options 参数。

text：选项按钮旁的文字。

font：字体。

height：选项按钮的文字有几行，默认是 1 行。

width：选项按钮的文字区间有几个字符宽，省略时会自行调整为实际宽度。

padx：默认是 1，可设置选项按钮与文字的间隔。

pady：默认是 1，可设置选项按钮的上下间距。

value：选项按钮的值，可以区分所选取的选项按钮。

indicatoron：当此值为 0 时，可以建立盒子选项按钮。

command：当用户更改选项时，会自动执行此函数。

variable：设置或取得目前选取的单选按钮，它的值类型通常是 IntVar 或 StringVar。

程序实例 ch18_24.py：这是一个简单选项按钮的应用，程序刚执行时默认选项是“男生”，此时窗口上方显示“尚未选择”，然后可以选择“男生”或“女生”，选择完成后可以显示“你是男生”或“你是女生”。

```
1  # ch18_24.py
2  from tkinter import *
3  def printSelection():
4      label.config(text="你是" + var.get())
5
6  window = Tk()
7  window.title("ch18_24")                          # 窗口标题
8
9  var = StringVar()
10 var.set("男生")                                   # 默认选项
11 label = Label(window,text="尚未选择", bg="lightyellow",width=30)
12 label.pack()
13
14 rb1 = Radiobutton(window,text="男生",
15                   variable=var,value='男生',
16                   command=printSelection).pack()
17 rb2 = Radiobutton(window,text="女生",
18                   variable=var,value='女生',
19                   command=printSelection).pack()
20
21 window.mainloop()
```

执行结果



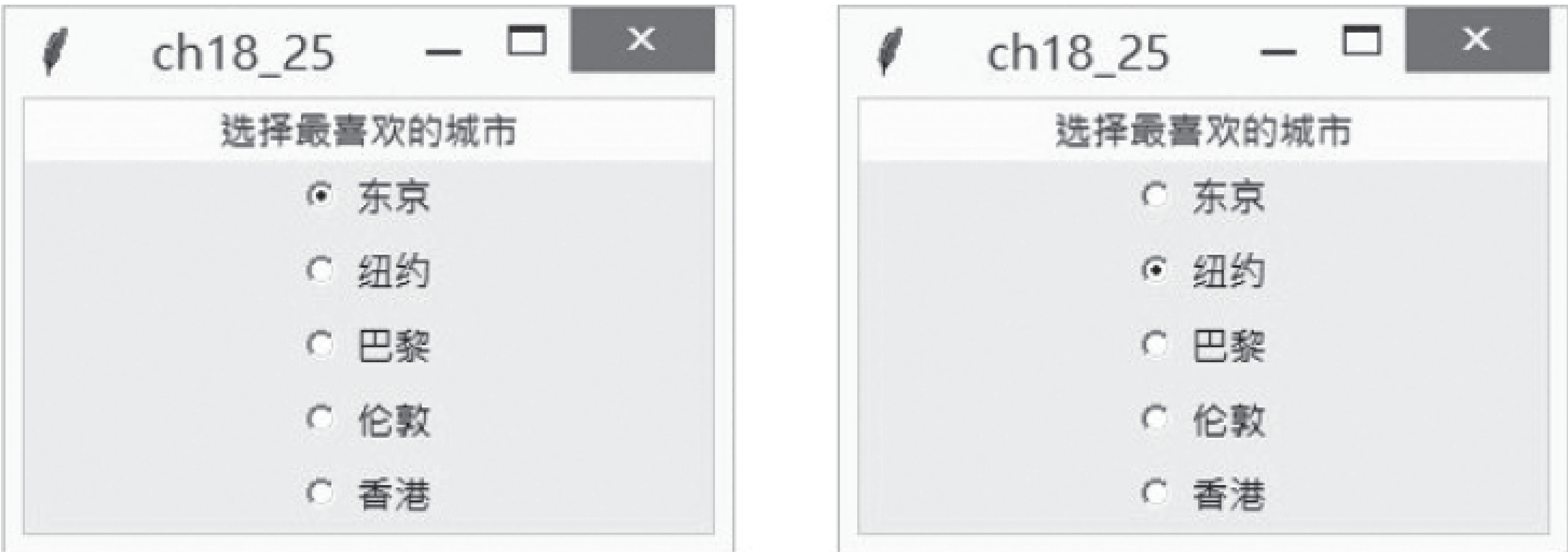
上述第 9 行是设置 var 变量是 StringVar( ) 对象，也是字符串对象。第 10 行是设置默认选项是“男生”，第 11 和 12 行是设置标签信息。第 14 ~ 16 行是建立“男生”选项按钮，第 17 ~ 19 行是建立“女生”选项按钮。当有按钮产生时，会执行第 3、4 行的函数，这个函数会由 var.get( ) 获得目前选项按钮，然后将此选项按钮对应的 value 值设置给标签对象 label 的 text，所以可以看到所选的结果。

上述建立选项按钮的方法虽然好用，但是当选项变多时程序就会显得比较复杂，此时可以考虑使用字典存储选项，然后用遍历字典方式建立选项按钮，可参考下列实例。

程序实例 ch18_25.py：为字典内的城市数据建立选项按钮，当我们选择最喜欢的城市时，Python Shell 窗口将列出所选的结果。

```
1  # ch18_25.py
2  from tkinter import *
3  def printSelection():
4      print(cities[var.get()])          # 列出所选城市
5
6  window = Tk()
7  window.title("ch18_25")               # 窗口标题
8  cities = {0:"东京",1:"纽约",2:"巴黎",3:"伦敦",4:"香港"}
9
10 var = IntVar()
11 var.set(0)                            # 默认选项
12 label = Label(window,text="选择最喜欢的城市",
13               fg="blue",bg="lightyellow",width=30).pack()
14
15 for val, city in cities.items():      # 建立选项按钮
16     Radiobutton(window,
17                 text=city,
18                 variable=var,value=val,
19                 command=printSelection).pack()
20
21 window.mainloop()
```

执行结果 下列左边是最初画面，右边是选择“纽约”。



当选择“纽约”选项按钮时，可以在 Python Shell 窗口中看到下列结果。

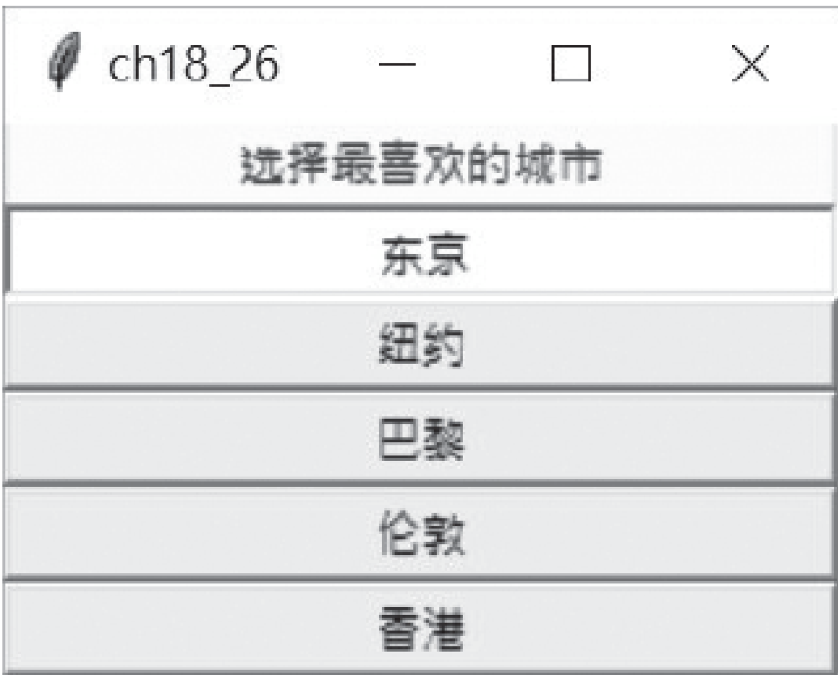
```
==================== RESTART: D:\Python\ch18\ch18_25.py ====================
纽约
```

此外，tkinter 也提供盒子选项按钮的概念，可以在 Radiobutton 方法内使用 indicatoron（意义是 indicator on）参数，将它设为 0。

程序实例 ch18_26.py：使用盒子选项按钮重新设计 ch18_25.py，重点是第 18 行。

```
15 for val, city in cities.items():      # 建立选项按钮
16     Radiobutton(window,
17                 text=city,
18                 indicatoron = 0,      # 用盒子取代选项按钮
19                 width=30,
20                 variable=var,value=val,
21                 command=printSelection).pack()
```

执行结果



18-10 复选框 Checkbutton

复选框在屏幕上是一个方框，它与选项按钮最大的差异在于它是复选。我们可以使用 Checkbutton( ) 方法建立复选框，它的使用方法如下。

Checkbutton(**父对象**, options, …)

Checkbutton( ) 方法的第一个参数是**父对象**，表示这个复选框将建立在哪一个窗口内。下列是 Checkbutton( ) 方法内其他常用的 options 参数。

text：复选框旁的文字。

font：字体。

height：复选框的文字有几行，默认是 1 行。

width：复选框的文字有几个字符宽，省略时会自行调整为实际宽度。

padx：默认是 1，可设置复选框与文字的间隔。

pady：预设是 1，可设置复选框的上下间距。

command：当用户更改选项时，会自动执行此函数。

variable：设置或取得目前选取的复选框，它的值类型通常是 IntVar 或 StringVar。

程序实例 ch18_27.py：建立复选框的应用。

```
1  # ch18_27.py
2  from tkinter import *
3
4  window = Tk()
5  window.title("ch18_27")                          # 窗口标题
6
7  Label(window,text="请选择喜欢的运动",
8          fg="blue",bg="lightyellow",width=30).grid(row=0)
9
10 var1 = IntVar()
11 Checkbutton(window,text="美式足球",
12                 variable=var1).grid(row=1,sticky=W)
13 var2 = IntVar()
14 Checkbutton(window,text="棒球",
15                 variable=var2).grid(row=2,sticky=W)
16 var3 = IntVar()
17 Checkbutton(window,text="篮球",
18                 variable=var3).grid(row=3,sticky=W)
19
20 window.mainloop()
```

执行结果 下方左图是程序执行初始画面，右图是笔者尝试勾选后的画面。



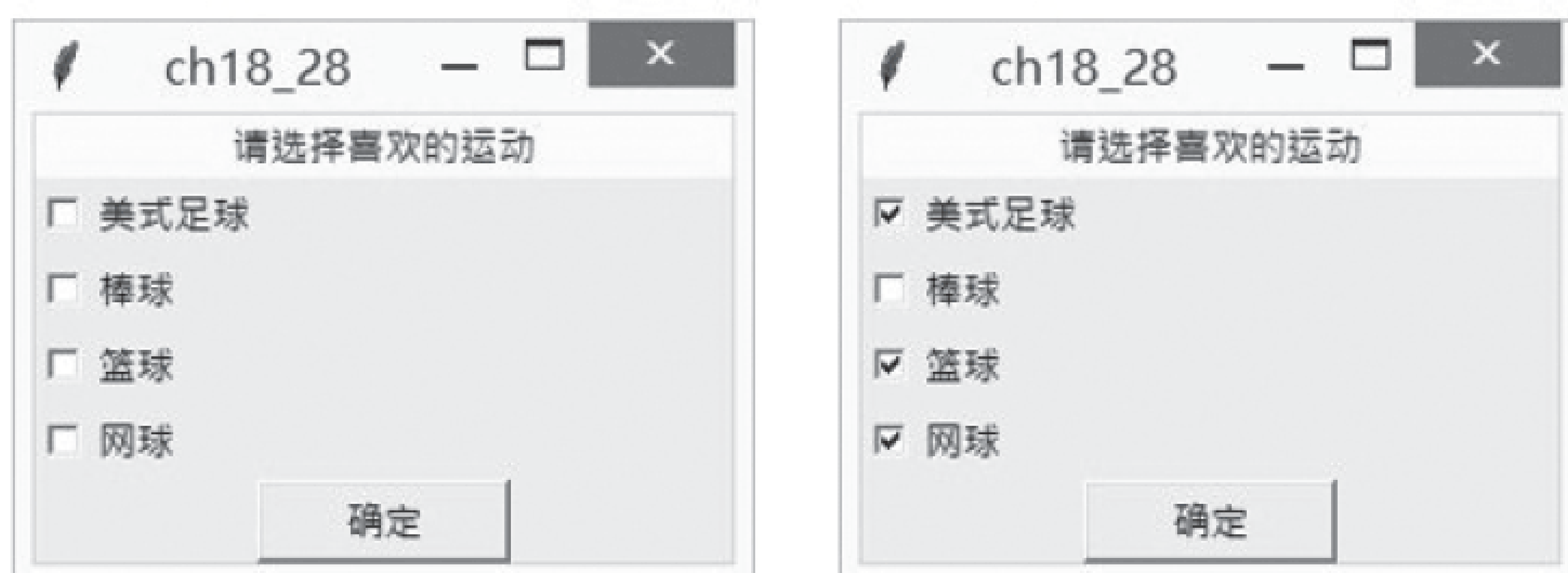
如果复选框项目不多时，可以参考上述实例使用 Checkbutton( ) 方法一步一步建立复选框的项目，如果项目很多时可以将项目组织成字典，然后使用循环观念建立选项，可参考下列实例。

程序实例 ch18_28.py：以 sports 字典方式存储运动复选框项目，然后建立此复选框，当有选择项

目时，若是单击“确定”按钮，可以在 Python Shell 窗口中列出所选的项目。

```
1  # ch18_28.py
2  from tkinter import *
3
4  def printInfo():
5      selection = ''
6      for i in checkboxes:                              # 检查此字典
7          if checkboxes[i].get() == True:               # 被选取则执行
8              selection = selection + sports[i] + "\t"
9      print(selection)
10
11 window = Tk()
12 window.title("ch18_28")                               # 窗口标题
13
14 Label(window,text="请选择喜欢的运动",
15       fg="blue",bg="lightyellow",width=30).grid(row=0)
16
17 sports = {0:"美式足球",1:"棒球",2:"篮球",3:"网球"}     # 运动字典
18 checkboxes = {}                                       # 字典存放被选取项目
19 for i in range(len(sports)):                          # 将运动字典转成复选框
20     checkboxes[i] = BooleanVar()                      # 布尔变量对象
21     Checkbutton(window,text=sports[i],
22                 variable=checkboxes[i]).grid(row=i+1,sticky=W)
23
24 Button(window,text="确定",width=10,command=printInfo).grid(row=i+2)
25
26 window.mainloop()
```

执行结果



上述右方若是单击“确定”按钮，可以在 Python Shell 窗口中看到下列结果。

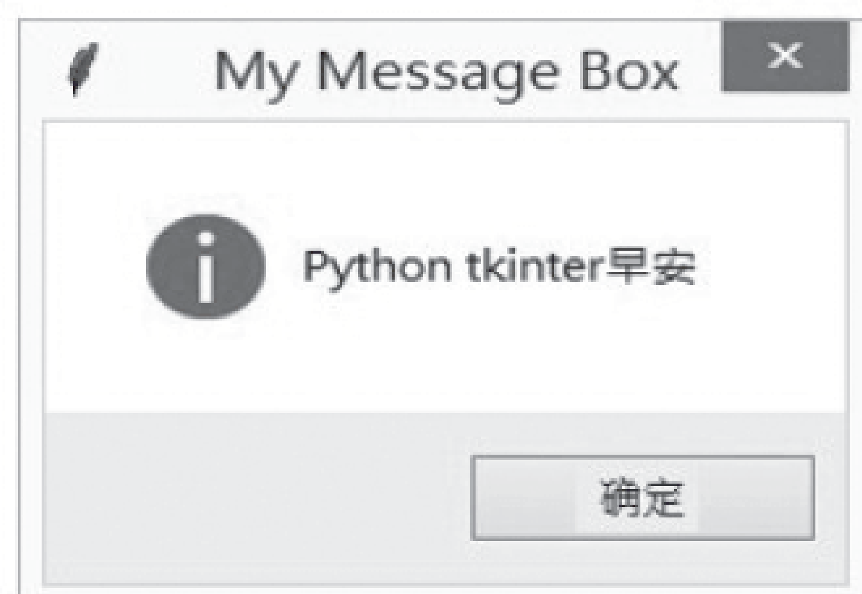
```
===================== RESTART: D:\Python\ch18\ch18_28.py =====================
美式足球        篮球    网球
```

上述第 17 行的 sports 字典是存储复选框的运动项目，第 18 行的 checkboxes 字典则是存储复选框是否被选取，第 19 ～ 22 行是循环将 sports 字典内容转成复选框，其中第 20 行是将 checkboxes 内容设为 BooleanVar 对象，经过这样设置未来第 7 行才可以用 get( ) 方法取得它的内容。第 24 行是建立“确定”按钮，当单击此按钮时会执行第 4 ～ 9 行的 printInfo( ) 函数，这个函数主要是将被选取的项目打印出来。

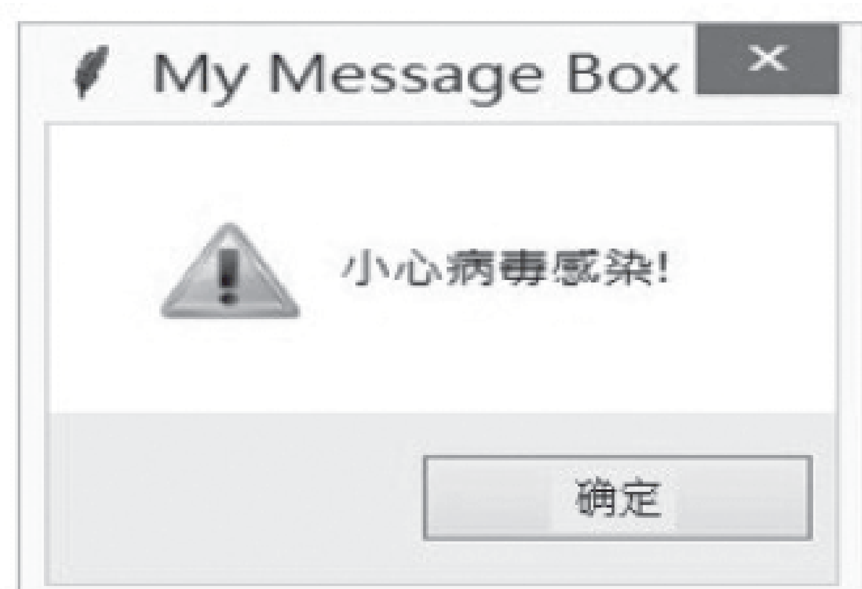
18-11 对话框 messagebox

Python 的 tkinter 模块内有 messagebox 模块，这个模块提供了 8 个对话框，这些对话框有不同的使用场合，本节将进行说明。

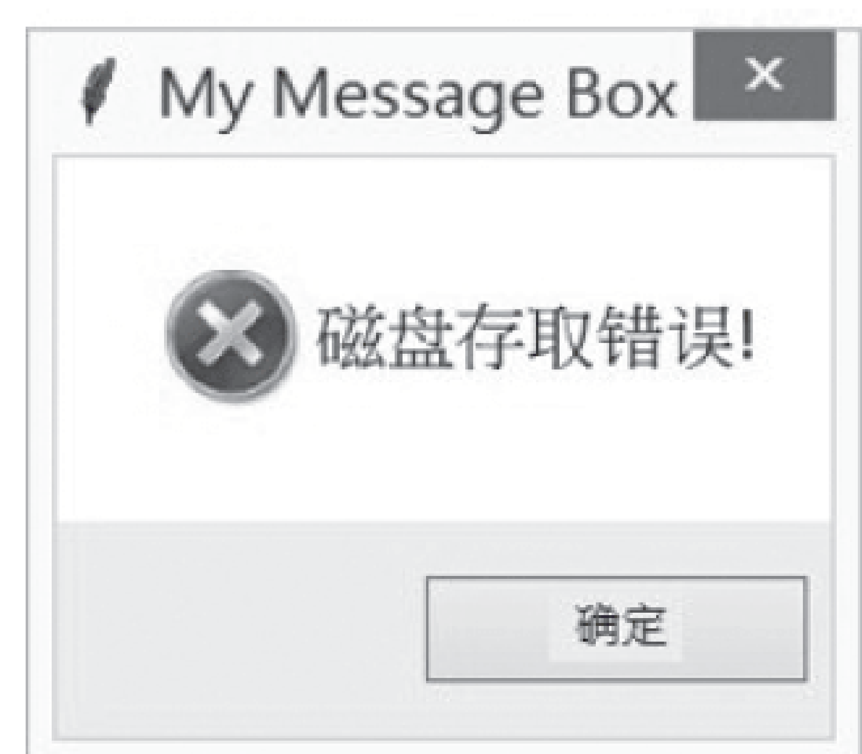
showinfo(title,message,options)：显示一般提示信息。



showwarning(title,message,options)：显示警告信息。



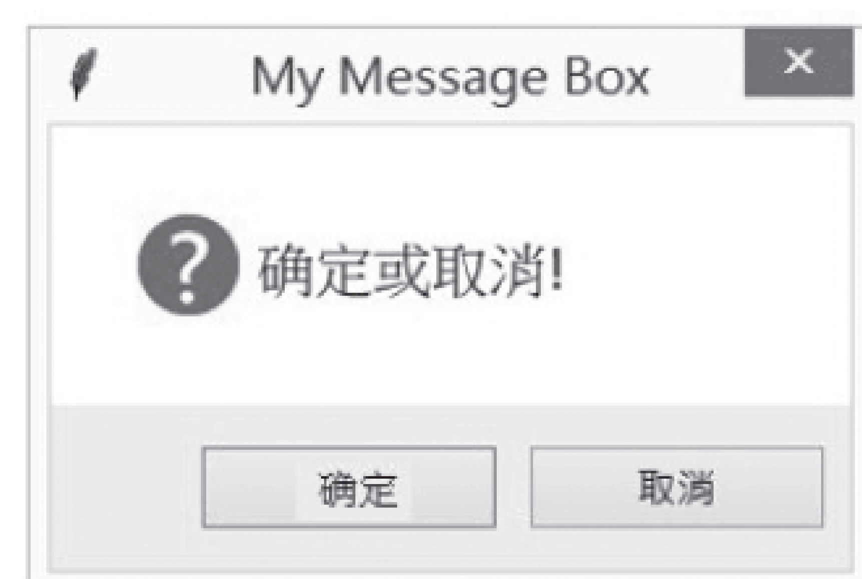
showerror(title,message,options)：显示错误信息。



askquestion(title,message,options)：显示询问信息。若单击“是”或 Yes 按钮会返回 yes，若单击“否”或 No 按钮会返回 no。



askokcancel(title,message,options)：显示确定或取消信息。若单击“确定”或 OK 按钮会返回 True，若单击“取消”或 Cancel 按钮会返回 False。

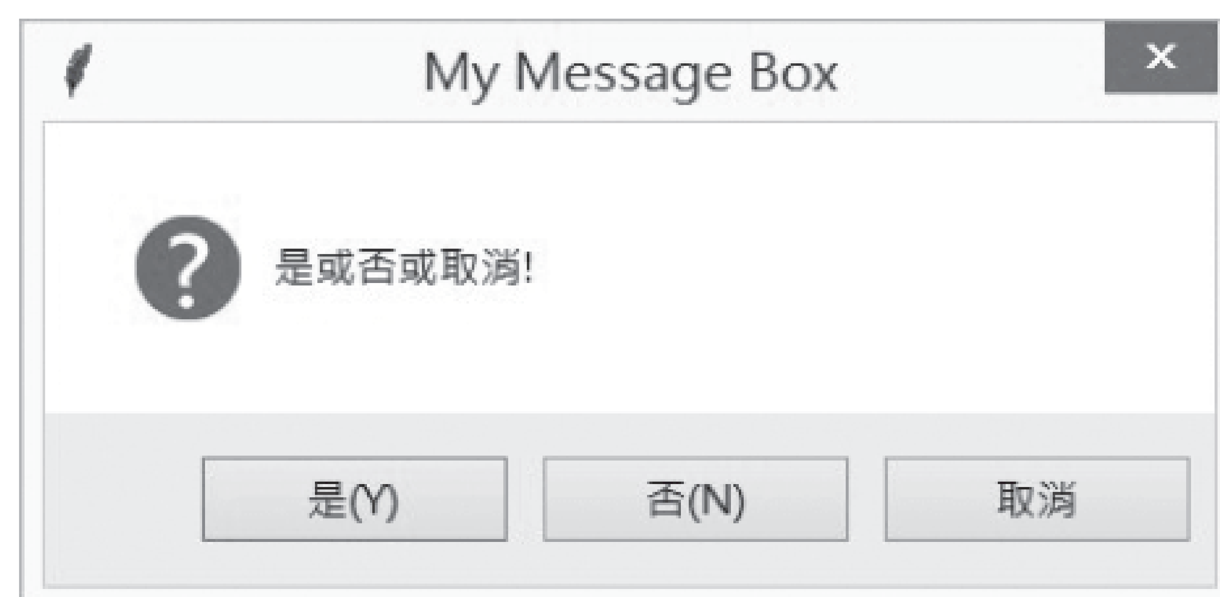


askyesno(title,message,options)：显示是或否信息。若单击“是”或 Yes 按钮会返回 True，若单

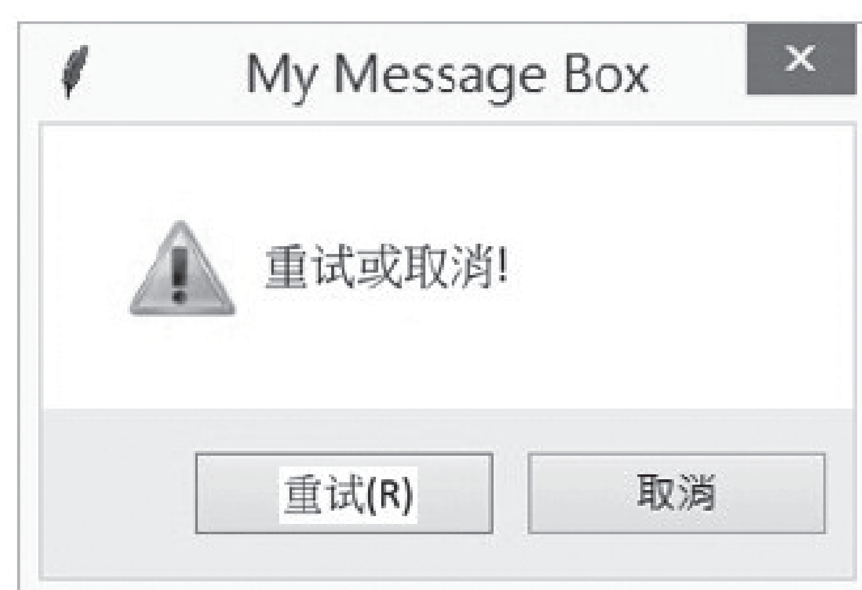
击“否”或 No 按钮会返回 False。



askyesnocancel(title,message,options)：显示是或否或取消信息。



askretrycancel(title,message,options)：显示重试或取消信息。若单击“重试”或 Retry 按钮会返回 True，若单击“取消”或 Cancel 按钮会返回 False。



上述对话框方法内的参数大致相同，title 是对话框的名称，message 是对话框内的文字。options 是选择性参数，可能有下列 3 种取值。

（1）default constant：默认按钮是 OK（确定）、Yes（是）、Retry（重试）在前面，也可更改此设置。

（2）icon(constant)：可设置所显示的图标，有 INFO、ERROR、QUESTION、WARNING 等 4 种图示可以设置。

（3）parent(widget)：指出当对话框关闭时，焦点窗口将返回父窗口。

程序实例 ch18_29.py：对话框的基本应用。

```
1  # ch18_29.py
2  from tkinter import *
3  from tkinter import messagebox
4
5  def myMsg():                      # 单击Good Morning按钮时执行
6      messagebox.showinfo("My Message Box","Python tkinter早安")
7
8  window = Tk()
9  window.title("ch18_29")           # 窗口标题
10 window.geometry("300x160")        # 窗口宽300高160
11
12 Button(window,text="Good Morning",command=myMsg).pack()
13
14 window.mainloop()
```

执行结果



18-12　图形 PhotoImage

图形功能可以应用于许多地方，例如，标签、功能按钮、选项按钮、文字区域等。在使用前可以用 PhotoImage( ) 方法建立此图形对象，然后再将此对象适度应用于其他窗口组件。它的语法如下。

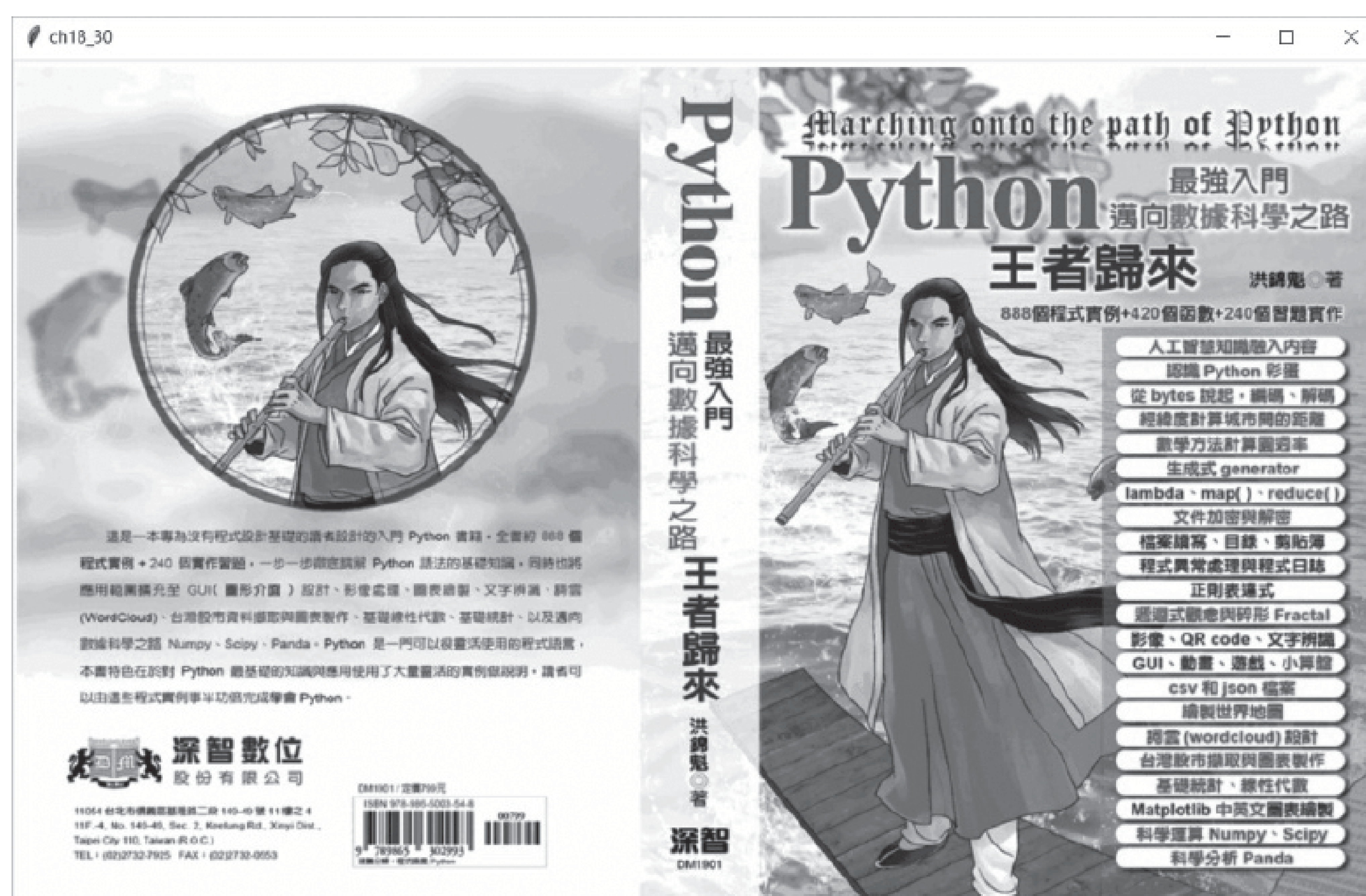
```
PhotoImage(file="xxx.gif")                # 扩展名 gif
```

需留意 PhotoImage( ) 方法早期只支持 gif 文件格式，不接受常用的 jpg 或 png 格式的文件，笔者发现目前已可以支持 png 文件了。建议将 gif 文件放在程序所在的文件夹。

程序实例 ch18_30.py：窗口显示 html.gif 文件的基本应用。

```
 1  # ch18_30.py
 2  from tkinter import *
 3
 4  window = Tk()
 5  window.title("ch18_30")          # 窗口标题
 6
 7  html_gif = PhotoImage(file="mybook.gif")
 8  Label(window,image=html_gif).pack()
 9
10  window.mainloop()
```

执行结果



18-12-1 图形与标签的应用

程序实例 ch18_31.py：窗口内同时有文字标签和图形标签的应用。

```
1  # ch18_31.py
2  from tkinter import *
3
4  window = Tk()
5  window.title("ch18_31")          # 窗口标题
6
7  sselogo = PhotoImage(file="sse.gif")
8  lab1 = Label(window,image=sselogo).pack(side="right")
9
10 sseText = """SSE全名是Silicon Stone Education,这家公司在美国,
11 这是国际专业认证公司,产品多元与丰富."""
12 lab2 = Label(window,text=sseText,bg="lightyellow",
13              padx=10).pack(side="left")
14
15 window.mainloop()
```

执行结果



由上图执行结果可以看到，文字标签第 2 行输出时，是默认居中对齐。可以在 Label( ) 方法内增加 justify=LEFT 参数，让第 2 行数据靠左输出。

程序实例 ch18_32.py：重新设计 ch18_31.py，让文字标签的第 2 行数据靠左输出，主要是第 13 行增加 justify=LEFT 参数。

```
12  lab2 = Label(window,text=sseText,bg="lightyellow",
13               justify=LEFT,padx=10).pack(side="left")
```

执行结果



18-12-2 图形与功能按钮的应用

一般功能按钮是用文字当作按钮名称，也可以用图形当作按钮名称，若要使用图形当作按钮，在 Button( ) 内可以省略 text 参数设置按钮名称，但是在 Button( ) 内要增加 image 参数设置图形对

象。若是要图形和文字并存在功能按钮上，需增加参数 compund=xx，xx 可以是 LEFT、TOP、RIGHT、BOTTOM、CENTER，分别代表图形在文字的左、上、右、下、中央。

程序实例 ch18_33.py：重新设计 ch18_12.py，使用 sun.gif 取代 Message 名称按钮。

```
1   # ch18_33.py
2   from tkinter import *
3
4   def msgShow():
5       label["text"] = "I love Python"
6       label["bg"] = "lightyellow"
7       label["fg"] = "blue"
8
9   window = Tk()
10  window.title("ch18_33")              # 窗口标题
11  label = Label(window)                # 标签内容
12
13  sun_gif = PhotoImage(file="sun.gif")
14  btn = Button(window,image=sun_gif,command=msgShow)
15
16  label.pack()
17  btn.pack()
18
19  window.mainloop()
```

执行结果



程序实例 ch18_33_1.py：将图像放在文字的上方，可参考上方第 3 张图。

```
14  btn = Button(window,image=sun_gif,command=msgShow,
15               text="Click me",compound=TOP)
```

程序实例 ch18_33_2.py：将图像放在文字的中央，可参考上方第 4 张图。

```
14  btn = Button(window,image=sun_gif,command=msgShow,
15               text="Click me",compound=CENTER)
```

18-13　尺度 Scale 的控制

Scale 可以翻译为尺度，Python 的 tkinter 模块有提供 Scale( ) 方法，我们可以移动尺度盒产生某一范围的数字。建立滚动条的方法是 Scale( )，它的语法格式如下。

Scale(**父对象**, options, …)

Scale( ) 方法的第一个参数是**父对象**，表示这个尺度控制将建立在哪一个窗口内。下列是 Scale( ) 方法内其他常用的 options 参数。

from_：尺度范围值的初值。

to：尺度范围值的末端值。

orient：默认是水平尺度，可以设置水平 HORIZONTAL 或垂直 VERTICAL。

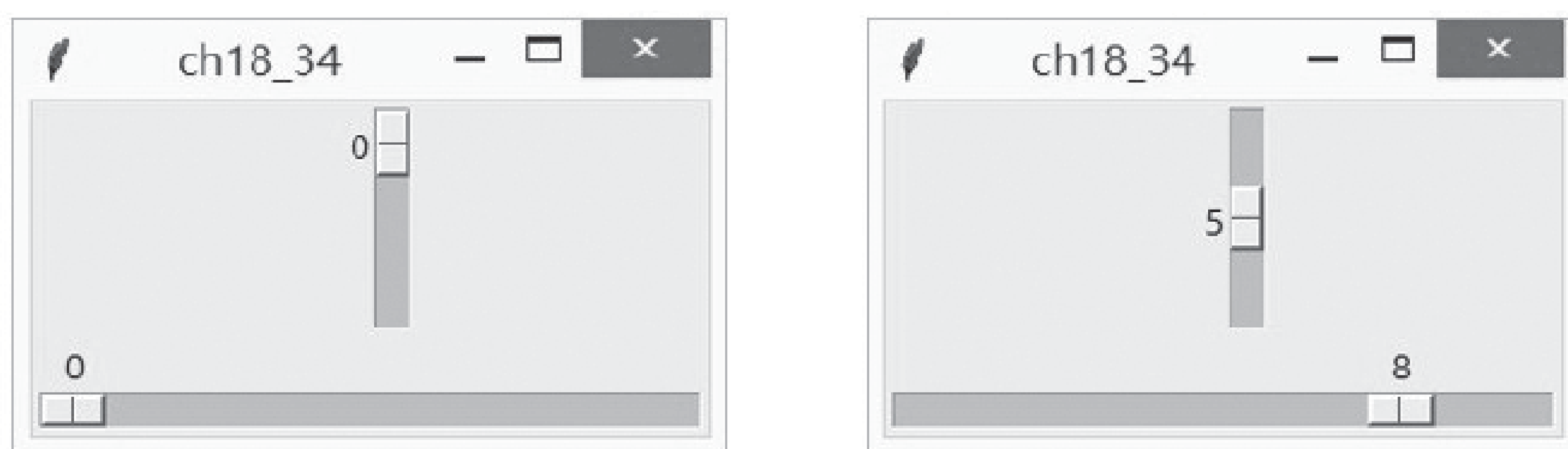
command：当用户更改选项时，会自动执行此函数。

length：尺度长度，默认是 100。

程序实例 ch18_34.py：一个简单的产生水平尺度与垂直尺度的应用，尺度值的范围为 0 ～ 10，垂直尺度使用默认长度，水平尺度则设为 300。

```
1  # ch18_34.py
2  from tkinter import *
3
4  window = Tk()
5  window.title("ch18_34")              # 窗口标题
6
7  slider1 = Scale(window,from_=0,to=10).pack()
8  slider2 = Scale(window,from_=0,to=10,
9                  length=300,orient=HORIZONTAL).pack()
10
11 window.mainloop()
```

执行结果



使用尺度时可以用 set( ) 方法设置尺度的值，用 get( ) 方法取得尺度的值。

程序实例 ch18_35.py：重新设计 ch18_34.py，这个程序会将水平尺度的初值设为 3，同时单击 Print 按钮可以在 Python Shell 窗口列出尺度值。

```
1  # ch18_35.py
2  from tkinter import *
3
4  def printInfo():
5      print(slider1.get(),slider2.get())
6
7  window = Tk()
8  window.title("ch18_35")              # 窗口标题
9
10 slider1 = Scale(window,from_=0,to=10)
11 slider1.pack()
12 slider2 = Scale(window,from_=0,to=10,
13                 length=300,orient=HORIZONTAL)
14 slider2.set(3)                       # 设置水平尺度值
15 slider2.pack()
16 Button(window,text="Print",command=printInfo).pack()
17
18 window.mainloop()
```

执行结果 下方左图是最初窗口，右图是调整后的结果。



在上述右图单击 Print 按钮后，可以得到下列尺度值的结果。

```
==================== RESTART: D:\Python\ch18\ch18_35.py ====================
5 7
```

18-14　菜单 Menu 的设计

窗口中一般会有菜单设计。菜单是一种下拉式的窗体，在这个窗体中可以设计菜单项。建立菜单的方法是 Menu( )，它的语法格式如下。

Menu(**父对象**，options，…)

Menu( ) 方法的第一个参数是**父对象**，表示这个菜单将建立在哪一个窗口内。下列是 Menu( ) 方法内其他常用的 options 参数。

activebackground：当鼠标移置此菜单项时的背景色。

bg：菜单项未被选取时的背景色。

fg：菜单项未被选取时的前景色。

image：菜单项的图示。

tearoff：菜单上方的分隔线，有分隔线时 tearoff 等于 1，此时菜单项从 1 开始放置。如果将 tearoff 设为 0 时，此时不会显示分隔线，但是菜单项将从 0 开始存放。

下列是其他相关的方法。

add_cascade( )：建立分层菜单，同时让此子功能项目与父菜单建立链接。

add_command( )：增加菜单项。

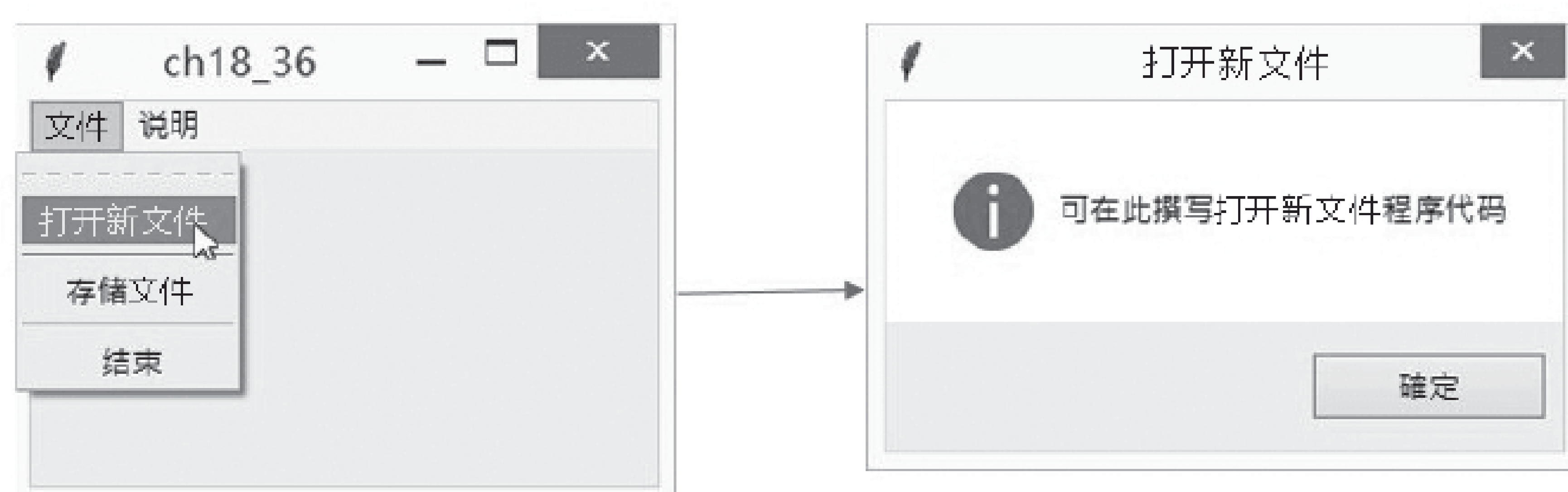
add_separator( )：增加分隔线。

程序实例 ch18_36.py：菜单的设计，这个程序设计了“文件”与“说明”菜单，在“文件”菜单内有“打开新文件”“存储文件”与“结束”菜单项。在“说明”菜单内有“程序说明”项目。

```
1  # ch18_36.py
2  from tkinter import *
3  from tkinter import messagebox
4
5  def newfile():
6      messagebox.showinfo("打开新文件","可在此撰写打开新文件程序代码")
7
8  def savefile():
9      messagebox.showinfo("存储文件","可在此撰写存储文件程序代码")
10
11 def about():
12     messagebox.showinfo("程序说明","作者:洪锦魁")
13
14 window = Tk()
15 window.title("ch18_36")
16 window.geometry("300x160")          # 窗口宽300高160
17
18 menu = Menu(window)                 # 建立菜单对象
19 window.config(menu=menu)
20
21 filemenu = Menu(menu)               # 建立“文件”菜单
22 menu.add_cascade(label="文件",menu=filemenu)
23 filemenu.add_command(label="打开新文件",command=newfile)
24 filemenu.add_separator()            # 增加分隔线
25 filemenu.add_command(label="存储文件",command=savefile)
26 filemenu.add_separator()            # 增加分隔线
```

```
27  filemenu.add_command(label="结束",command=window.destroy)
28
29  helpmenu = Menu(menu)              # 建立"说明"菜单
30  menu.add_cascade(label="说明",menu=helpmenu)
31  helpmenu.add_command(label="程序说明",command=about)
32
33  mainloop()
```

执行结果

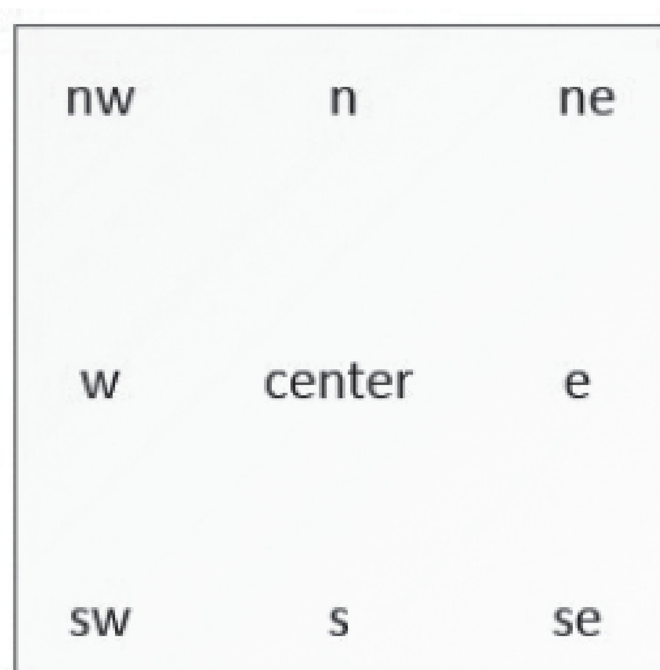


上述第 18、19 行是建立菜单对象。第 21 ～ 27 行是建立“**文件**”菜单，此菜单内有“**打开新文件**”“**存储文件**”“**结束**”菜单项，当执行“**打开新文件**”时会去执行第 5、6 行的 newfile( ) 函数，当执行“**存储文件**”时会去执行第 8、9 行的 savefile( ) 函数，当执行“**结束**”时会结束程序。

上述第 29 ～ 31 行是建立“**说明**”菜单，此菜单内有程序“**说明**”菜单项，当执行“**说明**”功能时会去执行第 11、12 行的 about( ) 函数。

18-15 专题——设计小计算器

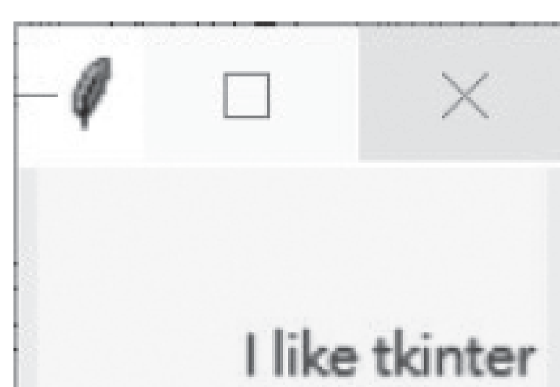
在此再介绍一个窗口控件的通用属性 anchor，所谓的锚（anchor）其实是指标签文字在标签区域输出位置的设置，在默认情况下 Widget 控件是上下与左右居中对齐，可以使用 anchor 选项设置组件的对齐方式。它的概念如下图。



程序实例 ch18_36_1.py：让字符串在标签右下方输出。

```
1   # ch18_36_1.py
2   from tkinter import *
3
4   root = Tk()
5   root.title("ch18_36_1")
6   label=Label(root,text="I like tkinter",
7               fg="blue",bg="yellow",
8               height=3,width=15,
9               anchor="se")
10  label.pack()
11
12  root.mainloop()
```

执行结果

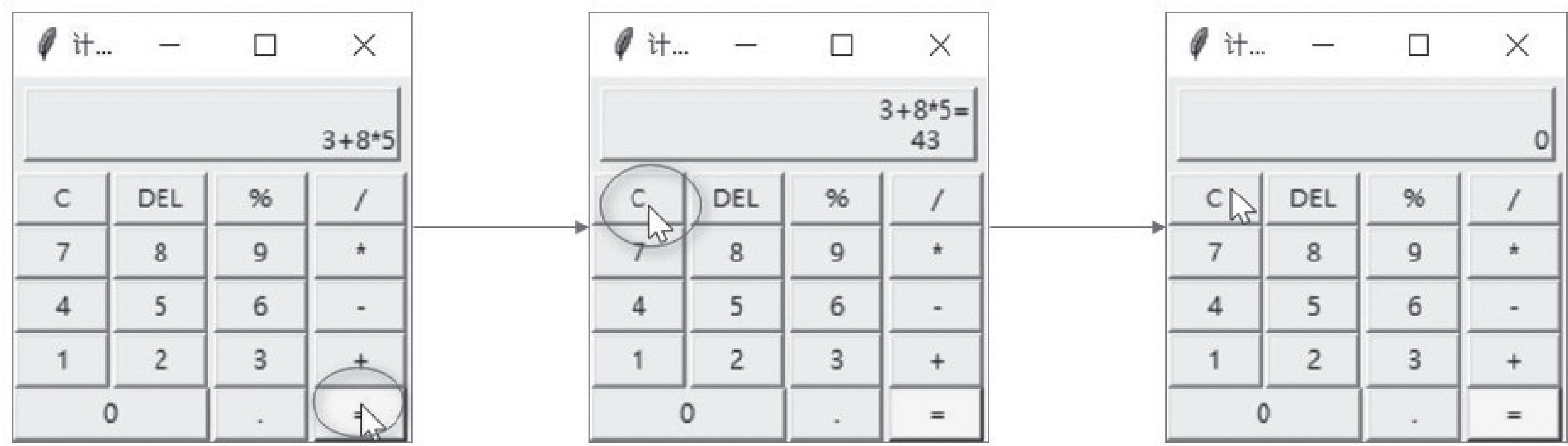


学会本章内容，其实就可以设计简单的小计算器了，下面将介绍完整的小计算器设计。

程序实例 ch18_37.py：设计简易的计算器，这个程序在按钮设计中大量使用 lambda，主要是数字按钮与算术表达式按钮使用相同的函数，只是传递的参数不一样，所使用的 lambda 可以简化设计。

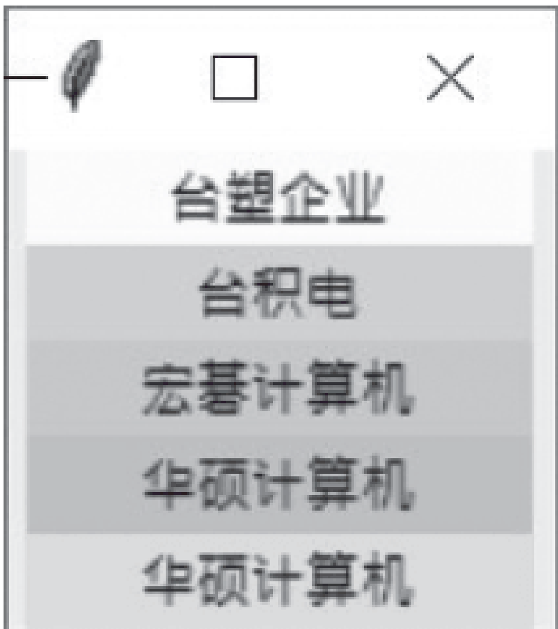
```
1  # ch18_37.py
2  from tkinter import *
3  def calculate():                          # 执行计算并显示结果
4      result = eval(equ.get())
5      equ.set(equ.get() + "=\n" + str(result))
6
7  def show(buttonString):                   # 更新显示区的计算公式
8      content = equ.get()
9      if content == "0":
10         content = ""
11     equ.set(content + buttonString)
12
13 def backspace():                          # 删除前一个字符
14     equ.set(str(equ.get()[:-1]))
15
16 def clear():                              # 清除显示区,放置0
17     equ.set("0")
18
19 root = Tk()
20 root.title("计算器")
21
22 equ = StringVar()
23 equ.set("0")                              # 默认是显示0
24
25 # 设计显示区
26 label = Label(root,width=25,height=2,relief="raised",anchor=SE,
27               textvariable=equ)
28 label.grid(row=0,column=0,columnspan=4,padx=5,pady=5)
29
30 # 清除显示区按钮
31 clearButton = Button(root,text="C",fg="blue",width=5,command=clear)
32 clearButton.grid(row = 1, column = 0)
33 # 以下是row1的其他按钮
34 Button(root,text="DEL",width=5,command=backspace).grid(row=1,column=1)
35 Button(root,text="%",width=5,command=lambda:show("%")).grid(row=1,column=2)
36 Button(root,text="/",width=5,command=lambda:show("/")).grid(row=1,column=3)
37 # 以下是row2的其他按钮
38 Button(root,text="7",width=5,command=lambda:show("7")).grid(row=2,column=0)
39 Button(root,text="8",width=5,command=lambda:show("8")).grid(row=2,column=1)
40 Button(root,text="9",width=5,command=lambda:show("9")).grid(row=2,column=2)
41 Button(root,text="*",width=5,command=lambda:show("*")).grid(row=2,column=3)
42 # 以下是row3的其他按钮
43 Button(root,text="4",width=5,command=lambda:show("4")).grid(row=3,column=0)
44 Button(root,text="5",width=5,command=lambda:show("5")).grid(row=3,column=1)
45 Button(root,text="6",width=5,command=lambda:show("6")).grid(row=3,column=2)
46 Button(root,text="-",width=5,command=lambda:show("-")).grid(row=3,column=3)
47 # 以下是row4的其他按钮
48 Button(root,text="1",width=5,command=lambda:show("1")).grid(row=4,column=0)
49 Button(root,text="2",width=5,command=lambda:show("2")).grid(row=4,column=1)
50 Button(root,text="3",width=5,command=lambda:show("3")).grid(row=4,column=2)
51 Button(root,text="+",width=5,command=lambda:show("+")).grid(row=4,column=3)
52 # 以下是row5的其他按钮
53 Button(root,text="0",width=12,
54        command=lambda:show("0")).grid(row=5,column=0,columnspan=2)
55 Button(root,text=".",width=5,
56        command=lambda:show(".")).grid(row=5,column=2)
57 Button(root,text="=",width=5,bg ="yellow",
58        command=lambda:calculate()).grid(row=5,column=3)
59
60 root.mainloop()
```

执行结果

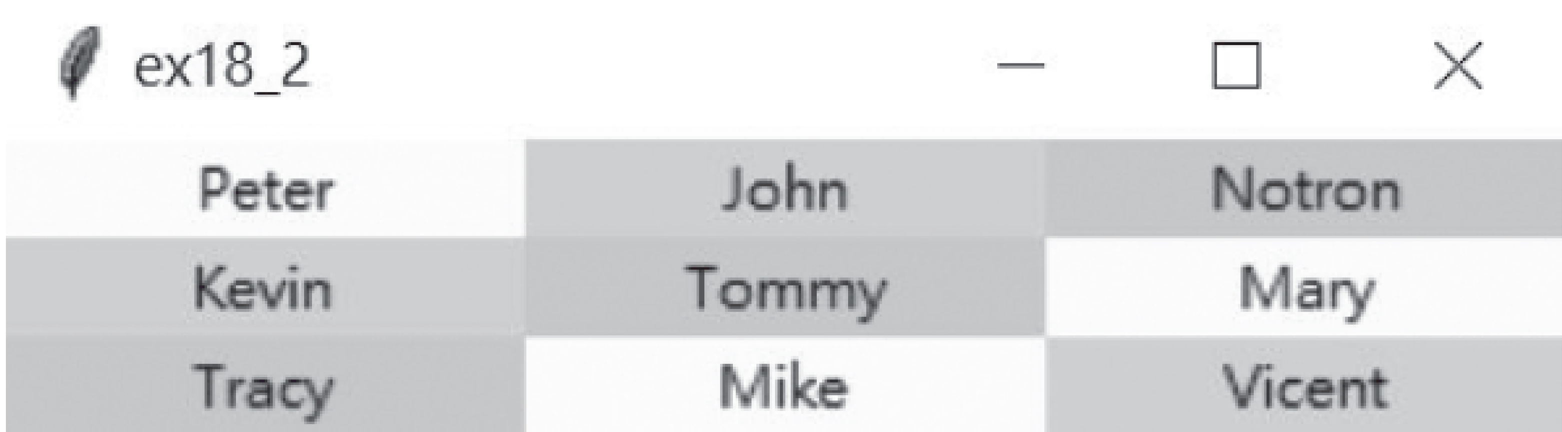


习题

1. 请参考 ch18_5.py，列出 5 个你心中敬佩的企业。(18-3 节)



2. 请参考 ch18_10.py，列出 9 个你心中的好朋友。



3. 请参考 ch18_20.py，将加法标签改成可以由按不同按钮修改的运算符号，同时增加加法、减法、乘法、除法功能按钮，当输入两个数字后，单击加、减、乘、除钮后可以单击等号按钮计算结果，这个程序同时需要自行设计整个窗口的组件配置。(18-6 节)



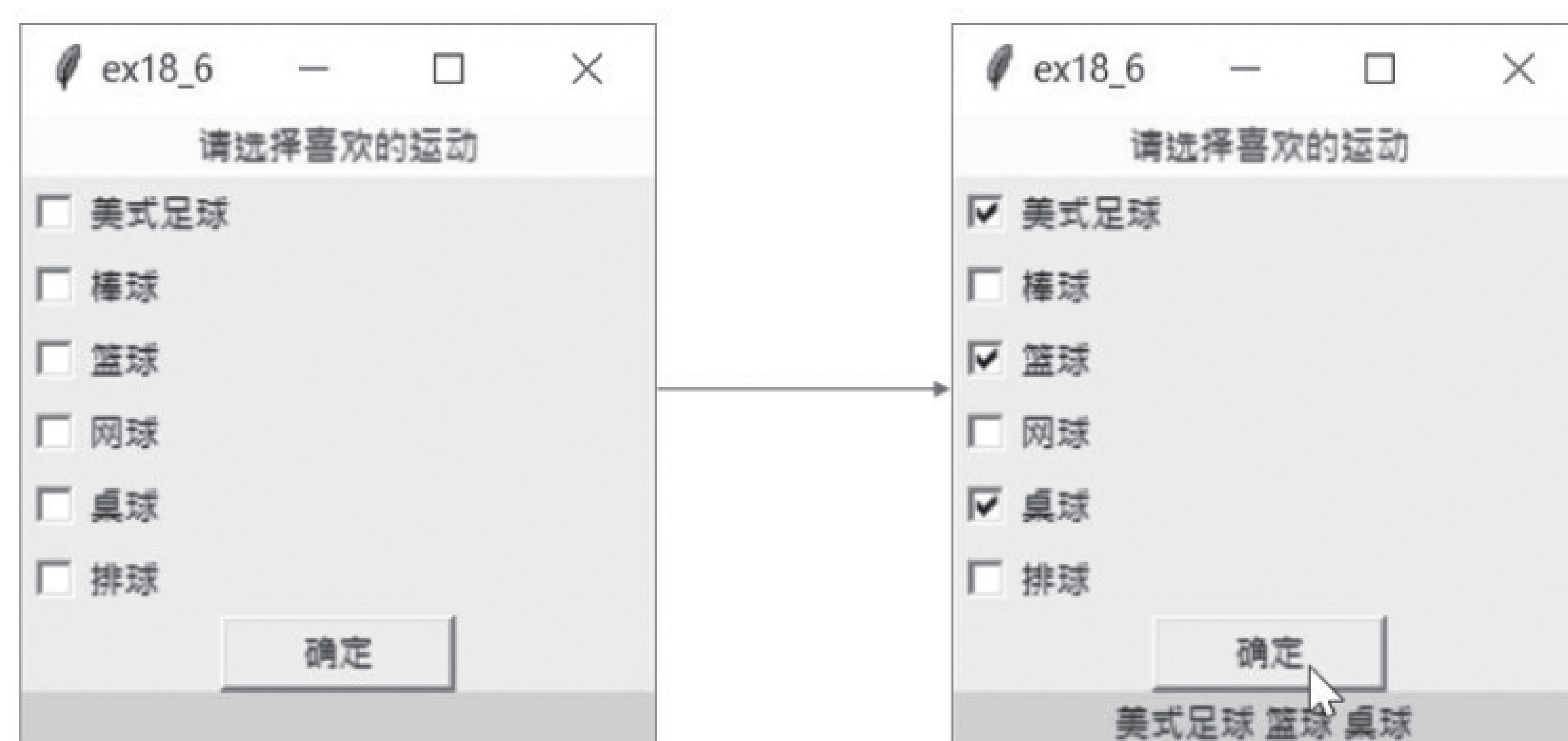
4. 贷款程序设计，本书程序实例 ch4_21.py 是一个房屋贷款程序，请使用 tkinter 重新设计此程序。这个程序的每月支付金额与总支付金额使用浅黄色为背景，未来我们可以输入利率、贷款年数、贷款金额然后计算每月支付金额与总支付金额，更多细节可以参考下列执行结果。



5. 请修改 ch18_25.py，请在下方增加设计标签，这个标签是浅绿色底色，程序执行之初是空白，当选择最喜欢的城市后可以在此标签中自动列出所选的城市。(18-9 节)



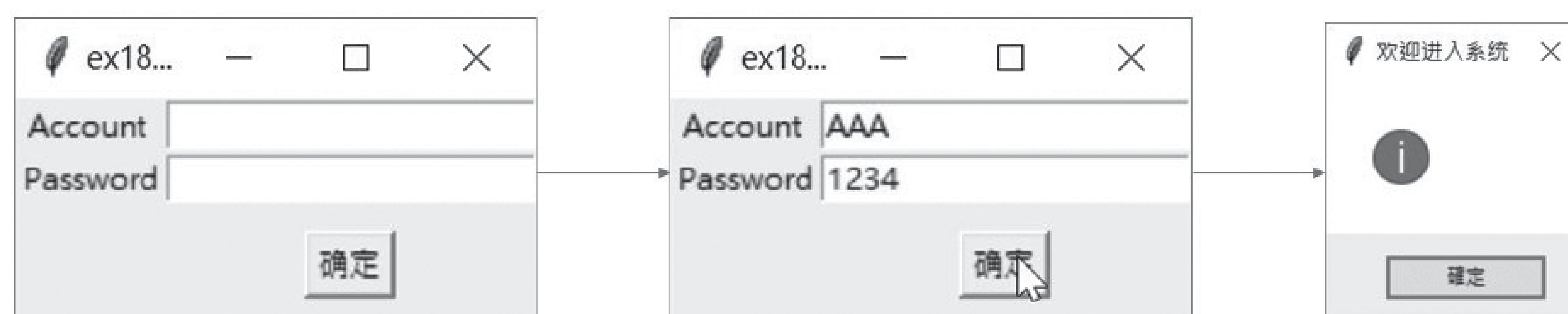
6. 请扩充设计 ch18_28.py，自行增加设计两种运动，同时在“确定”按钮下方增加浅绿色标签，当单击“确定”按钮后，可以在下方标签看到所选的运动，各运动间空一格。(18-10 节)

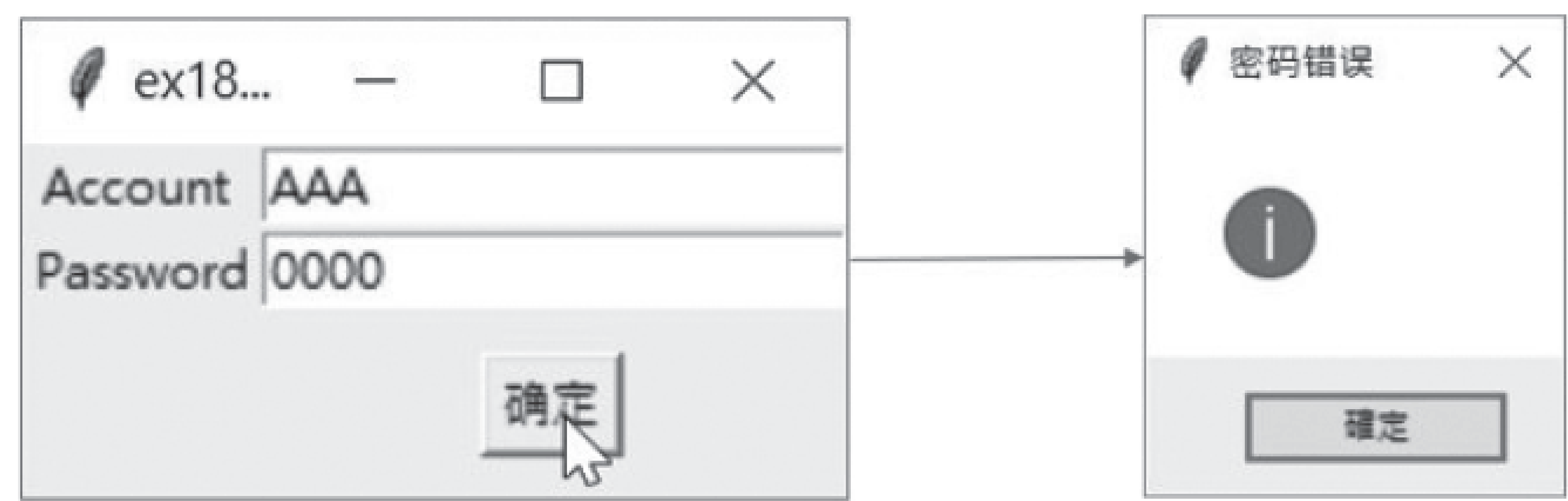


7. 请参考 ch18_17.py，但是功能按钮只有一个名称是“**确定**”，请在程序内建立一个字典，此字典内有 3 组账号和密码，如下所示。

```
accountDict = {"AAA":"1234", "BBB":"2345", "CCC":"3456"}
```

如果所输入的账号和密码正确，单击“**确定**”按钮时会出现“**欢迎进入系统**”的字符串提示对话框，如果输入账号错误会出现“**账号错误**”的警告对话框，如果输入密码错误会出现“**密码错误**”的警告对话框。(18-11 节)





8. 请参考 ch18_32.py，将图案改为自己的照片，同时写一段关于自己的叙述，此叙述必须至少有 3 行。（18-12 节）



9. 请参考 ch18_36.py，增加设计“编辑”菜单，此菜单内有“剪切”“复制”“粘贴”功能选项。（18-14 节）



19

第 19 章

动画与游戏

本章摘要

19-1 绘图功能
19-2 尺度控制画布背景颜色
19-3 动画设计
19-4 反弹球游戏设计
19-5 专题——使用 tkinter 处理谢尔宾斯基三角形

本章将介绍使用 Python 内建的模块 tkinter 制作动画，动画也是设计游戏的基础。

19-1 绘图功能

19-1-1 建立画布

可以使用 Canvas( ) 方法建立画布对象。

```
tk = Tk( )                                    # 使用 tk 当窗口 Tk 对象
canvas = Canvas(tk, width=xx, height=yy)      # xx,yy 是画布的宽与高
canvas.pack( )                                # 可以将画布包装好，这是必要的
```

画布建立完成后，左上角是坐标（0,0），x 轴向右递增，y 轴向下递增。

19-1-2 绘制线条 create_line( )

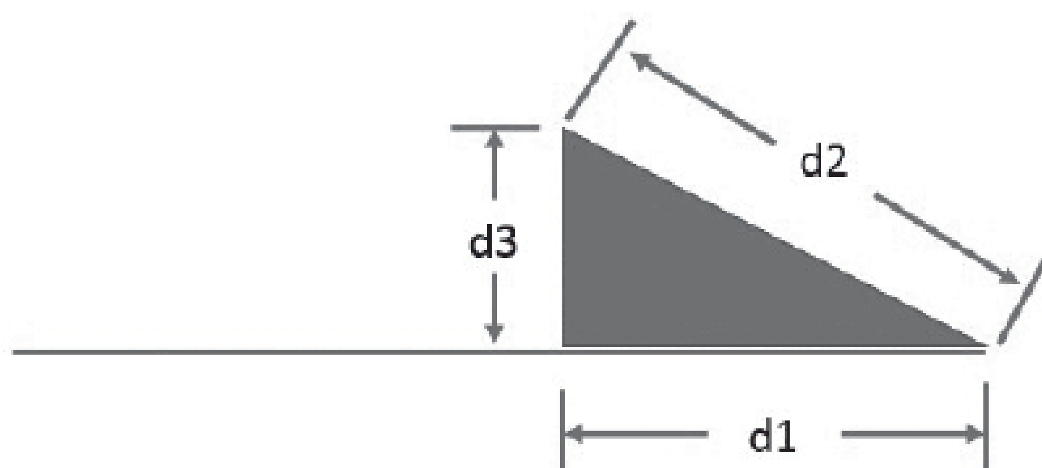
使用方式如下。

```
create_line(x1, y1, x2, y2, …, xn, yn, options)
```

线条将会沿着（x1,y1）,（x2,y2）, …绘制下去，下面是常用的 options 用法。

arrow：默认是没有箭头，使用 arrow=tk.FIRST 在起始线末端加上箭头，使用 arrow=tk.LAST 在最后一条线末端加上箭头，使用 arrow=tk.BOTH 在两端加上箭头。

arrowshape：使用元组（d1, d2, d3）代表箭头，默认是（8,10,3）。



capstyle：这是线条终点的样式，默认是 BUTT，也可以选择 PROJECTING、ROUND，程序实例可以参考 ch19_4.py。

dash：建立虚线，使用元组存储数字数据，第一个数字是实线，第二个数字是空白，如此循环，当所有元组数字用完又重新开始。例如，dash=（5,3）产生 5 像素实线，3 像素空白，如此循环。又如，dash=（8,1,1,1）产生 8 像素实线和点的线条，dash=（5,）产生 5 像素实线 5 像素空白。

dashoffset：与 dash 一样产生虚线，但是一开始数字是空白的宽度。

fill：设置线条颜色。

joinstyle：线条相交的设置，默认是 ROUND，也可以选择 BEVEL、MITER，程序实例可以参考 ch19_3.py。

stipple：绘制位图（Bitmap）线条，下面是在各操作系统平台可以使用的位图。程序实例可以参考 ch19_5.py。

```
error      hourglass   info      questhead   question
warning    gray12      gray25    gray50      gray75
```

下列是上述位图由左到右、由上到下依序的图例。



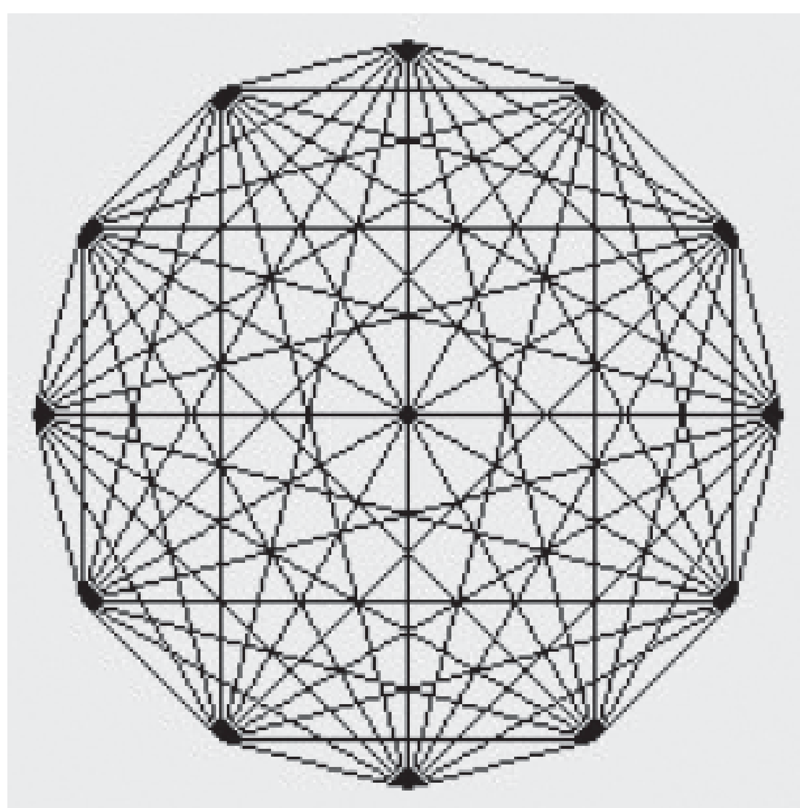
tags：为线条建立标签，未来配合使用 delete（删除标签），再重绘标签，可以创造动画效果，可参考 19-3-5 节。

width：线条宽度。

程序实例 ch19_1.py：在半径为 100 的圆外围建立 12 个点，然后将这些点彼此连接。

```
1   # ch19_1.py
2   from tkinter import *
3   import math
4
5   tk = Tk()
6   canvas = Canvas(tk, width=640, height=480)
7   canvas.pack()
8   x_center, y_center, r = 320, 240, 100
9   x, y = [], []
10  for i in range(12):           # 建立圆外围12个点
11      x.append(x_center + r * math.cos(30*i*math.pi/180))
12      y.append(y_center + r * math.sin(30*i*math.pi/180))
13  for i in range(12):           # 执行12个点彼此连接
14      for j in range(12):
15          canvas.create_line(x[i],y[i],x[j],y[j])
```

执行结果



上述程序使用了数学函数 sin( ) 和 cos( ) 以及 pi，这些是在 math 模块中。使用 create_line( ) 时，在 options 参数字段可以用 fill 设置线条颜色，用 width 设置线条宽度。

程序实例 ch19_2.py：不同线条颜色与宽度。

```
1   # ch19_2.py
2   from tkinter import *
3   import math
4
5   tk = Tk()
6   canvas = Canvas(tk, width=640, height=480)
7   canvas.pack()
8   canvas.create_line(100,100,500,100)
9   canvas.create_line(100,125,500,125,width=5)
10  canvas.create_line(100,150,500,150,width=10,fill='blue')
11  canvas.create_line(100,175,500,175,dash=(10,2,2,2))
```

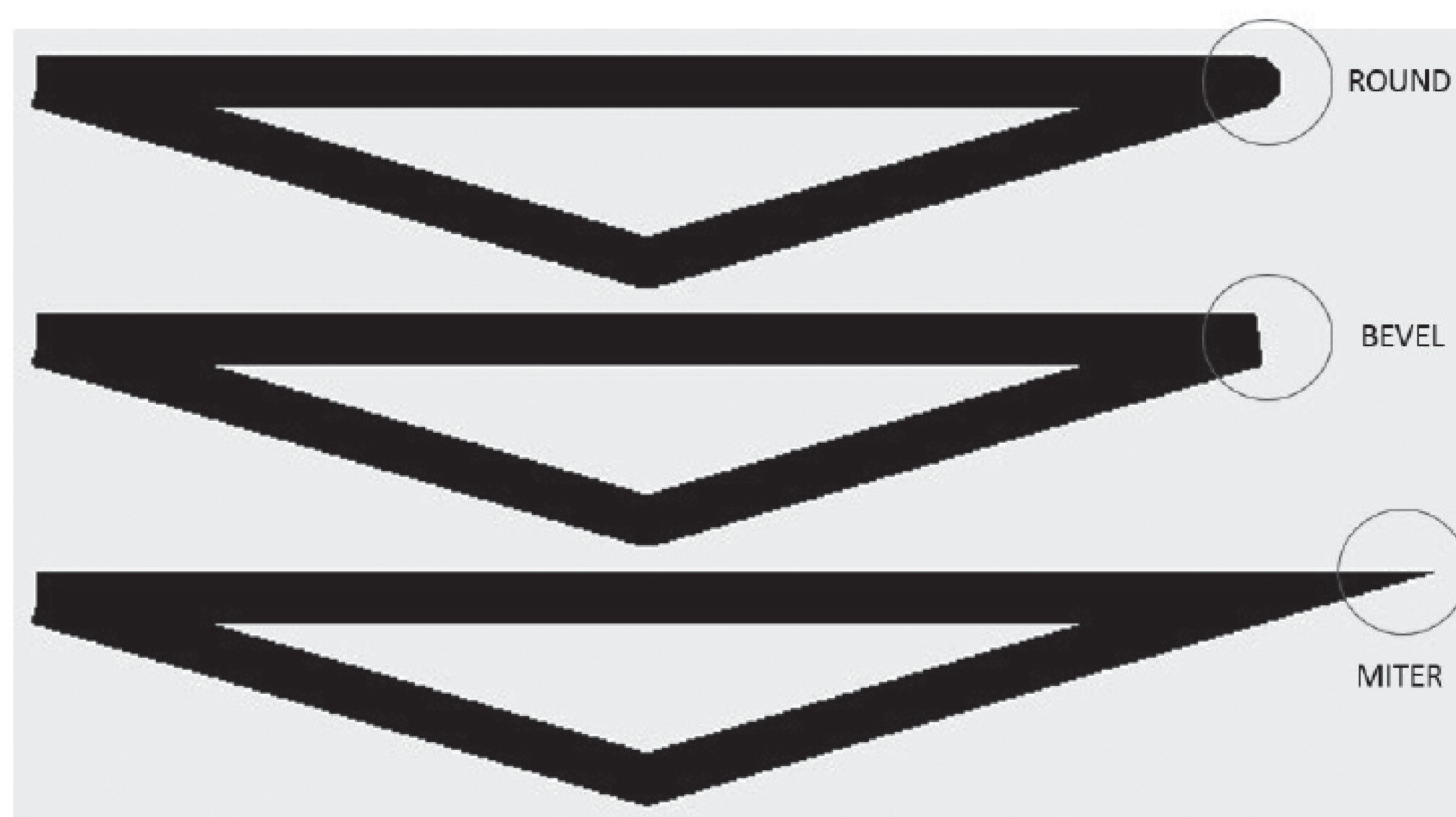
执行结果



程序实例 ch19_3.py：由线条交接了解 joinstyle 参数的应用。

```
1  # ch19_3.py
2  from tkinter import *
3  import math
4
5  tk = Tk()
6  canvas = Canvas(tk, width=640, height=480)
7  canvas.pack()
8  canvas.create_line(30,30,500,30,265,100,30,30,
9                     width=20,joinstyle=ROUND)
10 canvas.create_line(30,130,500,130,265,200,30,130,
11                    width=20,joinstyle=BEVEL)
12 canvas.create_line(30,230,500,230,265,300,30,230,
13                    width=20,joinstyle=MITER)
```

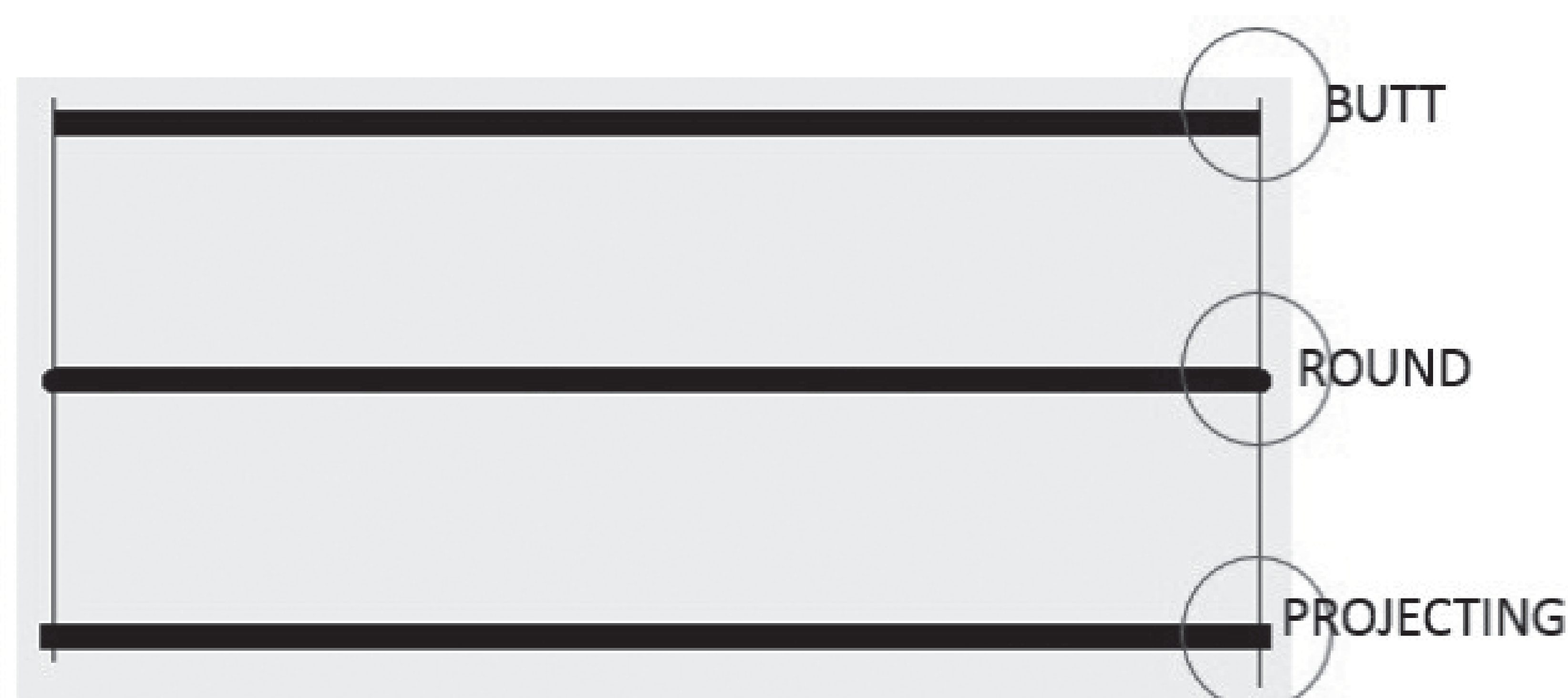
执行结果



程序实例 ch19_4.py：由线条了解 capstyle 参数的应用。

```
1  # ch19_4.py
2  from tkinter import *
3  import math
4
5  tk = Tk()
6  canvas = Canvas(tk, width=640, height=480)
7  canvas.pack()
8  canvas.create_line(30,30,500,30,width=10,capstyle=BUTT)
9  canvas.create_line(30,130,500,130,width=10,capstyle=ROUND)
10 canvas.create_line(30,230,500,230,width=10,capstyle=PROJECTING)
11 # 以下为垂直线
12 canvas.create_line(30,20,30,240)
13 canvas.create_line(500,20,500,250)
```

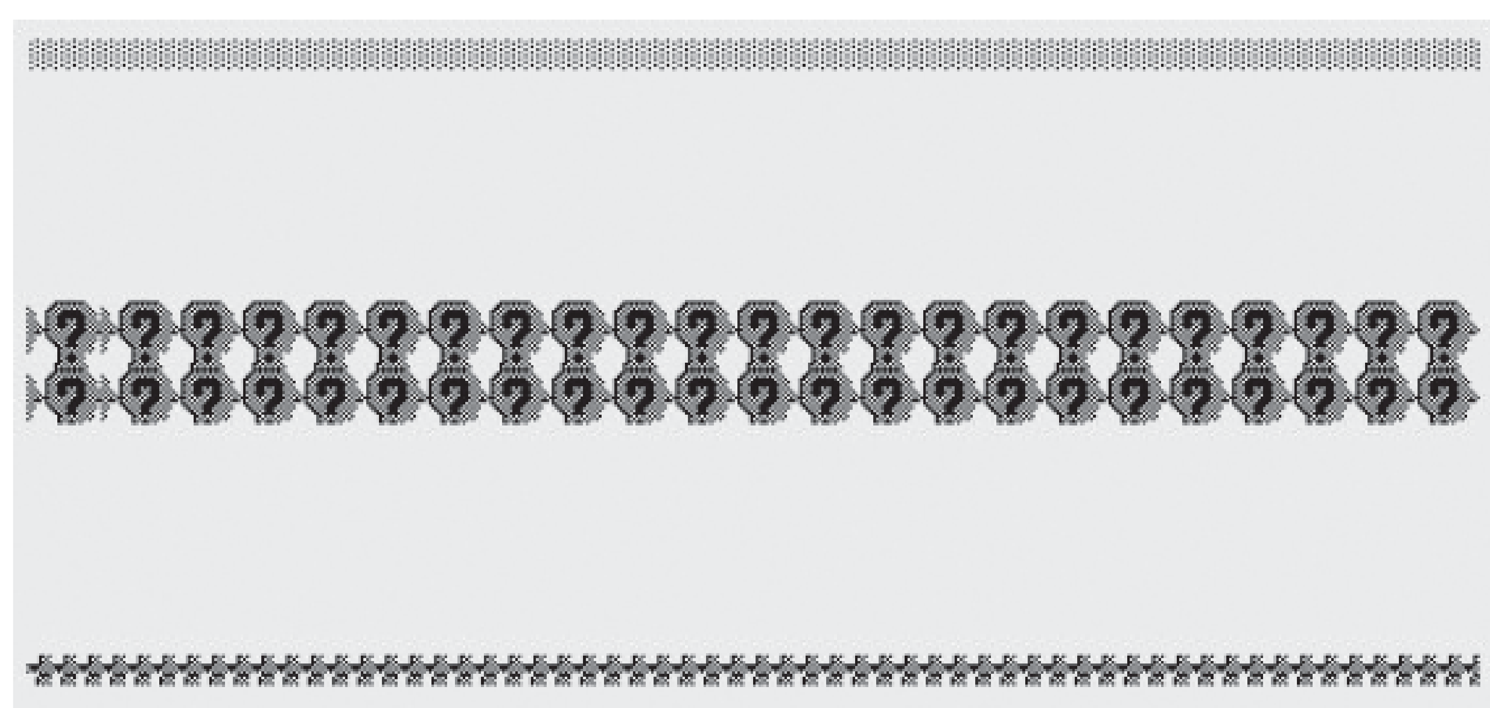
执行结果



程序实例 ch19_5.py：建立位图线条（stipple line）。

```
1  # ch19_5.py
2  from tkinter import *
3  import math
4
5  tk = Tk()
6  canvas = Canvas(tk, width=640, height=480)
7  canvas.pack()
8  canvas.create_line(30,30,500,30,width=10,stipple="gray25")
9  canvas.create_line(30,130,500,130,width=40,stipple="questhead")
10 canvas.create_line(30,230,500,230,width=10,stipple="info")
```

执行结果



19-1-3　绘制矩形 create_rectangle( )

使用方式如下。

create_rectangle(x1, y1, x2, y2,options)

（x1,y1）和（x2,y2）是矩形左上角和右下角坐标，下面是常用的 options 用法。

dash：建立虚线，概念与 create_line( ) 相同。

dashoffset：与 dash 一样产生虚线，但是一开始数字是空白的宽度。

fill：矩形填充颜色。

outline：设置矩形轮廓颜色。

stipple：绘制位图（Bitmap）矩形，可以参考 19-1-2 节，程序实例可以参考 ch19_5.py。

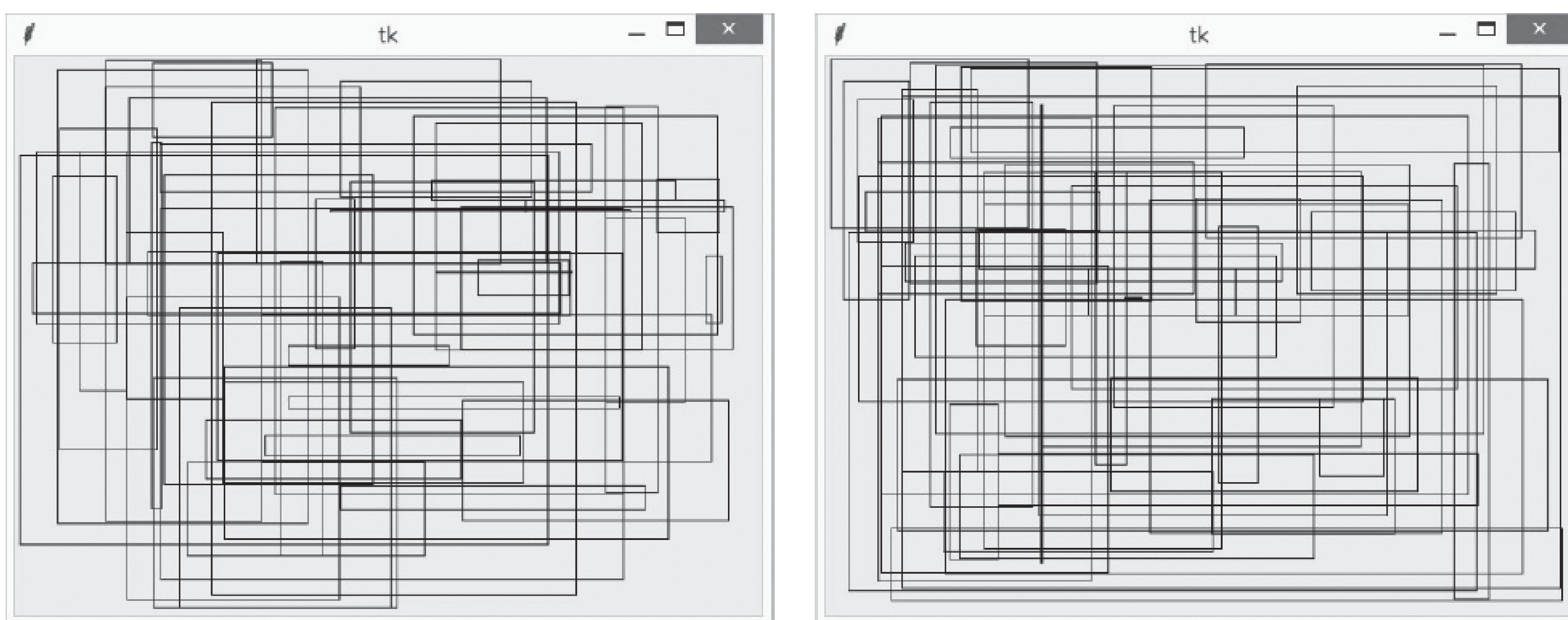
tags：为矩形建立标签，未来可以用 delete 创造动画效果，可参考 19-3-5 节。

width：矩形轮廓线宽度。

程序实例 ch19_6.py：在画布内随机产生不同位置与大小的矩形。

```
1  # ch19_6.py
2  from tkinter import *
3  from random import *
4
5  tk = Tk()
6  canvas = Canvas(tk, width=640, height=480)
7  canvas.pack()
8  for i in range(50):                     # 随机绘制50个不同位置与大小的矩形
9      x1, y1 = randint(1, 640), randint(1, 480)
10     x2, y2 = randint(1, 640), randint(1, 480)
11     if x1 > x2: x1,x2 = x2,x1          # 确保左上角x坐标小于右下角x坐标
12     if y1 > y2: y1,y2 = y2,y1          # 确保左上角y坐标小于右下角y坐标
13     canvas.create_rectangle(x1, y1, x2, y2)
```

执行结果



这个程序每次执行时都会产生不同的结果，有一点儿艺术画的效果。使用 create_rectangle( ) 时，在 options 参数字段可以用 fill='color' 设置矩形填充颜色，用 outline='color' 设置矩形轮廓颜色。

程序实例 ch19_7.py：绘制 3 个矩形，第一个使用红色填充，轮廓色是默认的；第二个使用黄色填充，轮廓是蓝色；第三个使用绿色填充，轮廓是灰色。

```
1  # ch19_7.py
2  from tkinter import *
3  from random import *
4
5  tk = Tk()
6  canvas = Canvas(tk, width=640, height=480)
7  canvas.pack()
8  canvas.create_rectangle(10, 10, 120, 60, fill='red')
9  canvas.create_rectangle(130, 10, 200, 80, fill='yellow', outline='blue')
10 canvas.create_rectangle(210, 10, 300, 60, fill='green', outline='grey')
```

执行结果



由执行结果可以发现，由于画布底色是浅灰色，所以第三个矩形用灰色轮廓，则几乎看不到轮廓线，另外也可以用 width 设置矩形轮廓的宽度。

19-1-4　绘制圆弧 create_arc( )

使用方式如下。

create_arc(**x1, y1, x2, y2,** extent=angle, style=ARC, **options**)

（x1,y1）和（x2,y2）分别是包围圆形矩形**左上角**和**右下角**的坐标，下面是常用的 options 用法。

dash：建立虚线，概念与 create_line( ) 相同。

dashoffset：与 dash 一样产生虚线，但是一开始数字是空白的宽度。

extent：表示圆弧范围，值介于 1 ～ 359。如果写 360 会视为 0。

fill：填充圆弧颜色。

outline：设置圆弧线条颜色。

start：圆弧起点位置。

stipple：绘制位图（Bitmap）圆弧。

style：有 3 种格式，ARC、CHORD、PIESLICE，可参考 ch19_9.py。

tags：为圆弧建立标签，未来可以用 delete 创造动画效果，可参考 19-3-5 节。

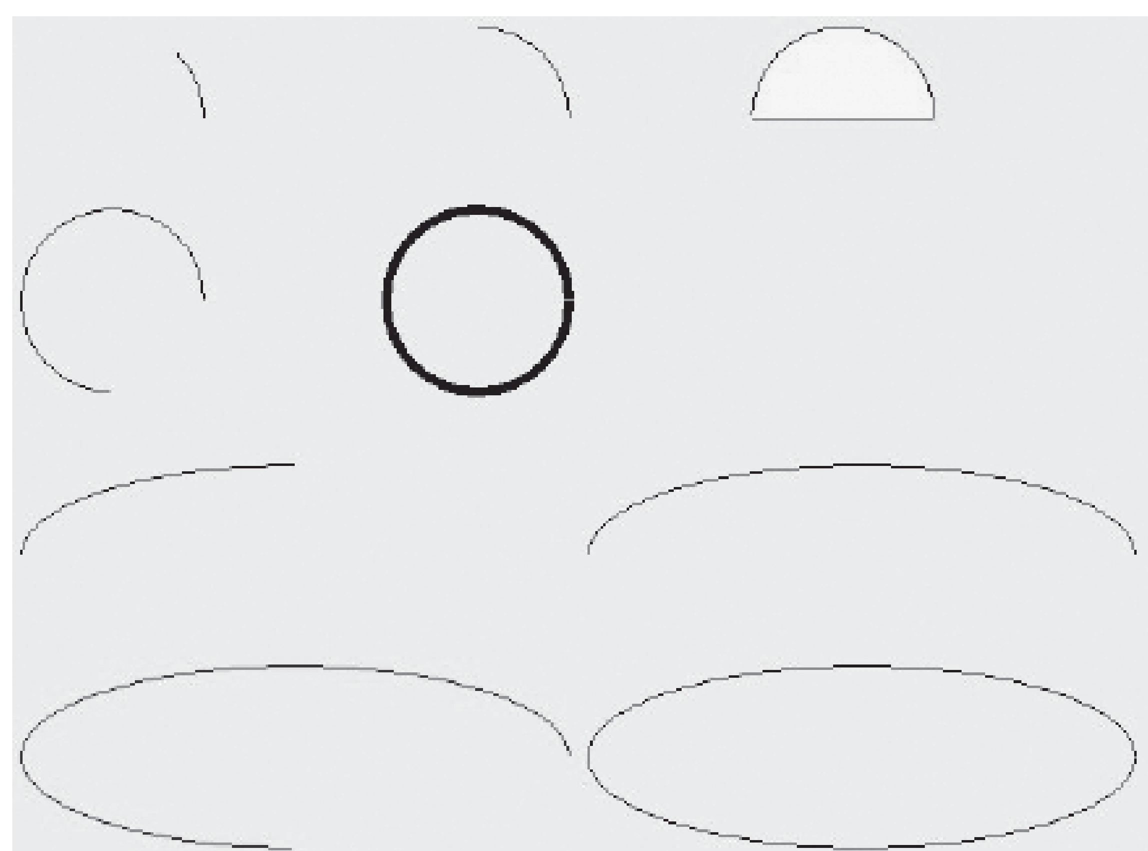
width：圆弧线条宽度。

上述 style=ARC 表示绘制圆弧，如果是要使用 options 参数填满圆弧则需舍去此参数。此外，options 参数可以使用 width 设置轮廓线条宽度（可参考 ch19_8.py 第 12 行），outline 设置轮廓线条颜色（可参考 ch19_8.py 第 16 行），fill 设置填充颜色（可参考 ch19_8.py 第 10 行）。目前默认绘制圆弧的起点是右边，也可以用 start=0 代表，也可以通过设置 start 的值更改圆弧的起点，方向是逆时针，可参考 ch19_8.py 第 14 行。

程序实例 ch19_8.py：绘制各种不同的圆和椭圆，以及圆弧和椭圆弧。

```
1  # ch19_8.py
2  from tkinter import *
3
4  tk = Tk()
5  canvas = Canvas(tk, width=640, height=480)
6  canvas.pack()
7  # 以下以圆形为基础
8  canvas.create_arc(10, 10, 110, 110, extent=45, style=ARC)
9  canvas.create_arc(210, 10, 310, 110, extent=90, style=ARC)
10 canvas.create_arc(410, 10, 510, 110, extent=180, fill='yellow')
11 canvas.create_arc(10, 110, 110, 210, extent=270, style=ARC)
12 canvas.create_arc(210, 110, 310, 210, extent=359, style=ARC, width=5)
13 # 以下以椭圆形为基础
14 canvas.create_arc(10, 250, 310, 350, extent=90, style=ARC, start=90)
15 canvas.create_arc(320, 250, 620, 350, extent=180, style=ARC)
16 canvas.create_arc(10, 360, 310, 460, extent=270, style=ARC, outline='blue')
17 canvas.create_arc(320, 360, 620, 460, extent=359, style=ARC)
```

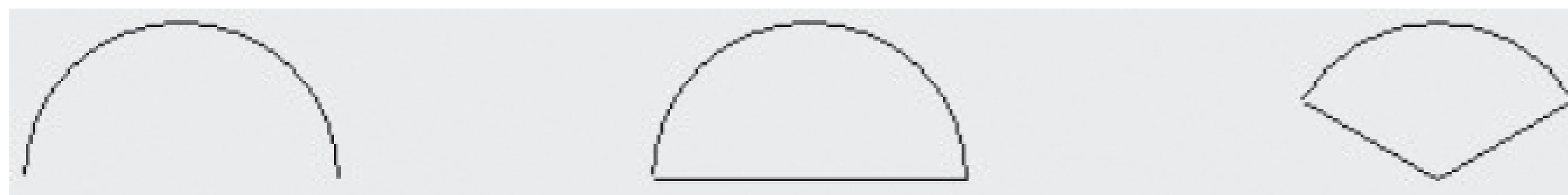
执行结果



程序实例 ch19_9.py：style 参数是 ARC、CHORD、PIESLICE 参数的应用。

```
1  # ch19_9.py
2  from tkinter import *
3
4  tk = Tk()
5  canvas = Canvas(tk, width=640, height=480)
6  canvas.pack()
7  # 以下以圆形为基础
8  canvas.create_arc(10, 10, 110, 110, extent=180, style=ARC)
9  canvas.create_arc(210, 10, 310, 110, extent=180, style=CHORD)
10 canvas.create_arc(410, 10, 510, 110, start=30, extent=120, style=PIESLICE)
```

执行结果



19-1-5 绘制圆或椭圆 create_oval( )

使用方式如下：

create_oval(x1, y1, x2, y2, options)

(x1,y1) 和 (x2,y2) 分别是包围圆形矩形左上角和右下角的坐标，下面是常用的 options 用法。

dash：建立虚线，概念与 create_line( ) 相同。

dashoffset：与 dash 一样产生虚线，但是一开始数字是空白的宽度。

fill：设置圆或椭圆的填充颜色。

outline：设置圆或椭圆的轮廓颜色

stipple：绘制位图（Bitmap）轮廓的圆或椭圆。

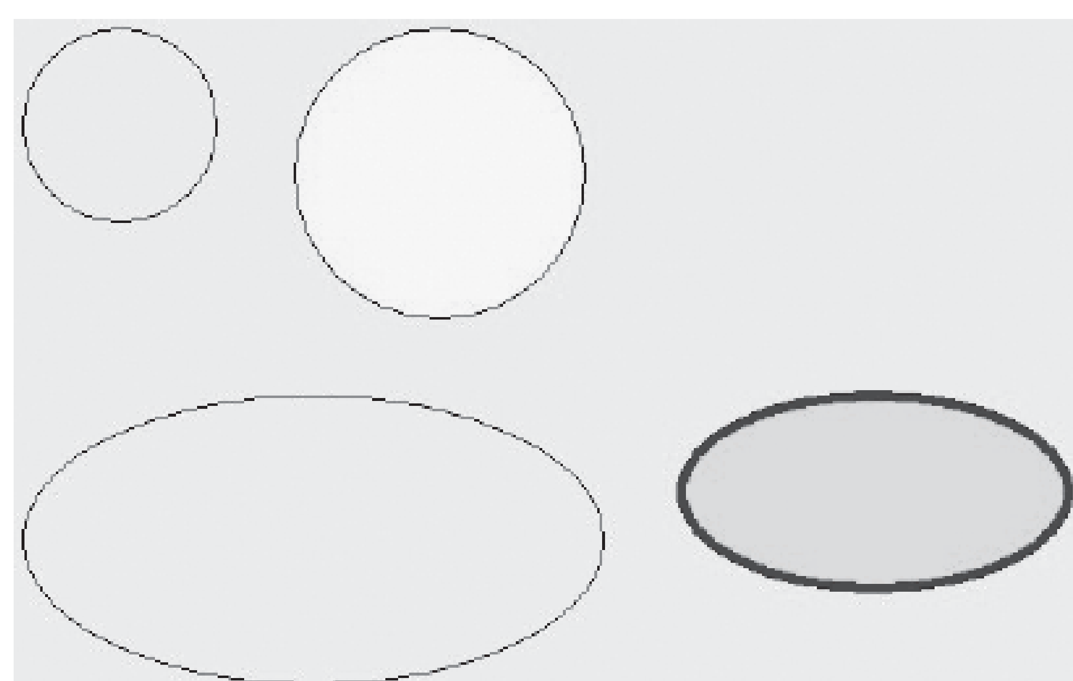
tags：为圆建立标签，未来可以用 delete 创造动画效果，可参考 19-3-5 节。

width：圆或椭圆轮廓线宽度。

程序实例 ch19_10.py：圆和椭圆的绘制。

```
1  # ch19_10.py
2  from tkinter import *
3
4  tk = Tk()
5  canvas = Canvas(tk, width=640, height=480)
6  canvas.pack()
7  # 以下是圆形
8  canvas.create_oval(10, 10, 110, 110)
9  canvas.create_oval(150, 10, 300, 160, fill='yellow')
10 # 以下是椭圆形
11 canvas.create_oval(10, 200, 310, 350)
12 canvas.create_oval(350, 200, 550, 300, fill='aqua', outline='blue', width=5)
```

执行结果



19-1-6　绘制多边形 create_polygon( )

使用方式如下。

create_polygon(x1, y1, x2, y2, x3, y3, … xn, yn, **options**)

(x1,y1) , … (xn,yn) 是多边形各角的（x,y）坐标，下面是常用的 options 用法。

dash：建立虚线，概念与 create_line( ) 相同。

dashoffset：与 dash 一样产生虚线，但是一开始数字是空白的宽度。

fill：设置多边形的填充颜色。

outline：设置多边形的轮廓颜色。

stipple：绘制位图（Bitmap）轮廓的多边形。

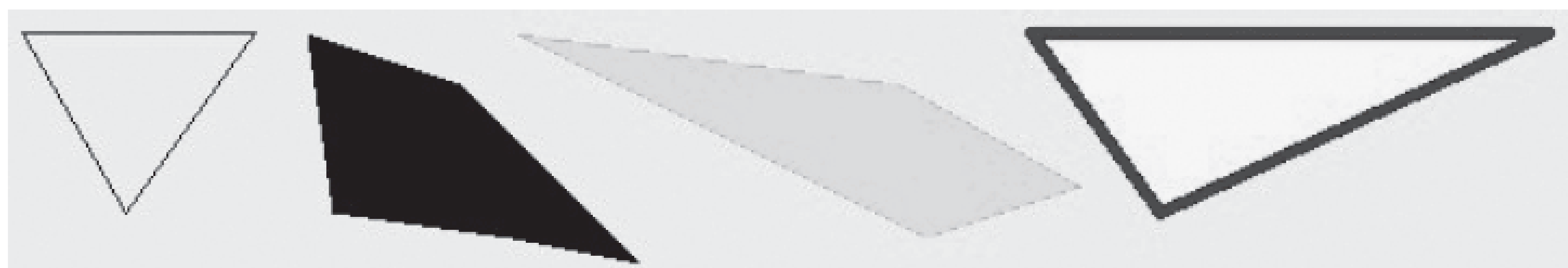
tags：为多边形建立标签，未来可以用 delete 创造动画效果，可参考 19-3-5 节。

width：多边形轮廓线宽度。

程序实例 ch19_11.py：绘制多边形的应用。

```
1  # ch19_11.py
2  from tkinter import *
3
4  tk = Tk()
5  canvas = Canvas(tk, width=640, height=480)
6  canvas.pack()
7  canvas.create_polygon(10,10, 100,10, 50,80, fill='', outline='black')
8  canvas.create_polygon(120,10, 180,30, 250,100, 200,90, 130,80)
9  canvas.create_polygon(200,10, 350,30, 420,70, 360,90, fill='aqua')
10 canvas.create_polygon(400,10,600,10,450,80,width=5,outline='blue',fill='yellow')
```

执行结果



19-1-7　输出文字 create_text( )

使用方式如下。

create_text(x,y,text=字符串 , **options**)

默认 (x,y) 是文字符串输出的中心坐标，下面是常用的 options 用法。

anchor：默认是 anchor=CENTER，也可以参考 18-5 节的位置概念。

fill：文字颜色。

font：字体的使用，可以参考 18-2 节。

justify：当输出多行时，默认是靠左 LEFT，可以参考 18-2 节。

stipple：绘制位图（Bitmap）线条的文字，默认是 "" 表示实线。

text：输出的文字。

tags：为文字建立标签，未来可以用 delete 创造动画效果，可参考 19-3-5 节。

程序实例 ch19_12.py：输出文字的应用。

```
1  # ch19_12.py
2  from tkinter import *
3
4  tk = Tk()
5  canvas = Canvas(tk, width=640, height=480)
6  canvas.pack()
7  canvas.create_text(200, 50, text='Ming-Chi Institute of Technology')
8  canvas.create_text(200, 80, text='Ming-Chi Institute of Technology', fill='blue')
9  canvas.create_text(300, 120, text='Ming-Chi Institute of Technology', fill='blue',
10                    font=('Old English Text MT',20))
11 canvas.create_text(300, 160, text='Ming-Chi Institute of Technology', fill='blue',
12                    font=('华康新综艺体 Std W7',20))
13 canvas.create_text(300, 200, text='明志科技大学', fill='blue',
14                    font=('华康新综艺体 Std W7',20))
```

执行结果



19-1-8 更改画布背景颜色

在使用 Canvas( ) 方法建立画布时，可以加上 bg 参数建立画布背景颜色。

程序实例 ch19_13.py：将画布背景改成黄色。

```
1  # ch19_13.py
2  from tkinter import *
3
4  tk = Tk()
5  canvas = Canvas(tk, width=640, height=240, bg='yellow')
6  canvas.pack()
```

执行结果



19-1-9 插入图像 create_image( )

在 Canvas 控件内可以使用 create_image( ) 在 Canvas 对象内插入图像文件，它的语法如下。

```
create_image(x, y, options)
```

（x,y）是图像左上角的位置，下面是常用的 options 用法。

anchor：默认是 anchor=CENTER，也可以参考 18-5 节的位置概念。

image：插入的图像。

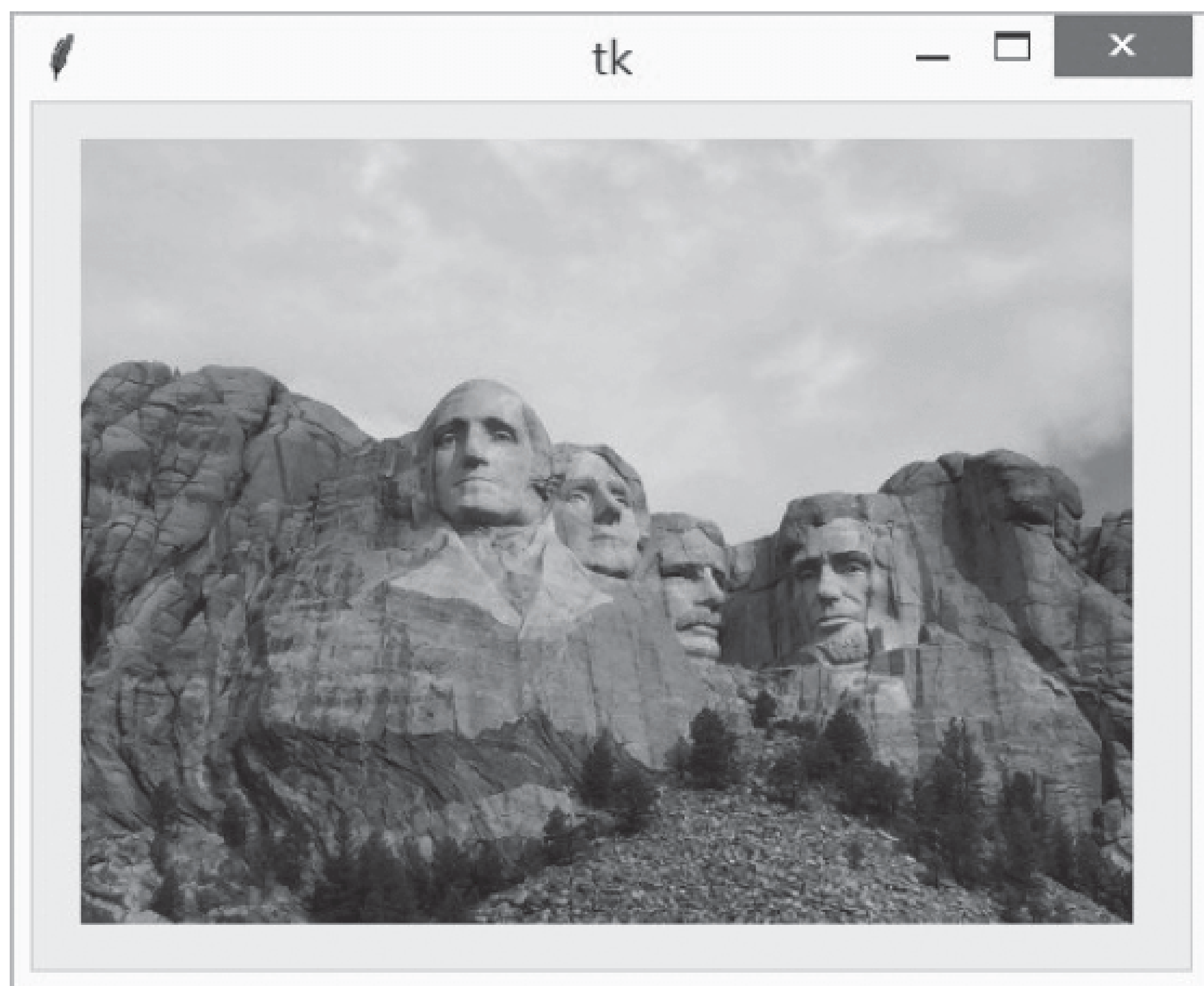
tags：为图像建立标签，未来可用 delete 创造动画效果，可参考 19-3-5 节。

下面将以实例解说。

程序实例 ch19_14.py：插入图像文件 rushmore.jpg，这个程序会建立窗口，在 x 轴方向大于图像宽度 30 像素，y 轴方向大于图像宽度 20 像素。

```
1  # ch19_14.py
2  from tkinter import *
3  from PIL import Image, ImageTk
4
5  tk = Tk()
6  img = Image.open("rushmore.jpg")
7  rushMore = ImageTk.PhotoImage(img)
8
9  canvas = Canvas(tk, width=img.size[0]+40,
10                 height=img.size[1]+30)
11 canvas.create_image(20,15,anchor=NW,image=rushMore)
12 canvas.pack(fill=BOTH,expand=True)
```

执行结果



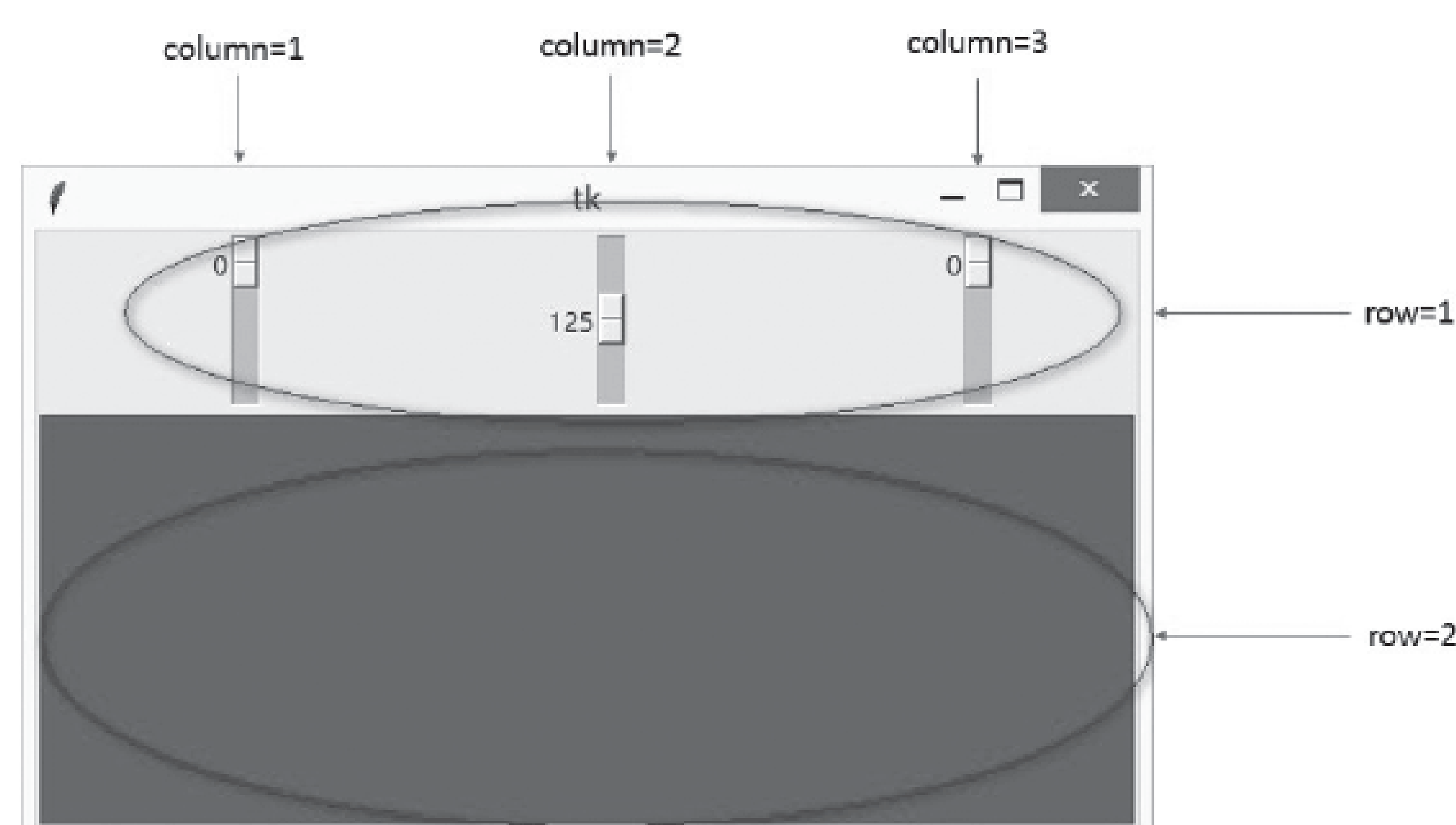
19-2　尺度控制画布背景颜色

第 18 章有介绍 tkinter 模块的尺度 Scale( )，利用这个方法可以获得尺度的值，下面将会利用 3 个尺度控制色彩的 R、G、B 值，然后可以控制画布背景颜色。

程序实例 ch19_15.py：使用尺度控制画布背景颜色，其中，为了让读者了解设置尺度初值的方法，第 17 行特别设置 gSlider 的尺度初值为 125。这个程序在执行时，若是有卷动尺度将调用 bfUpdate(source) 函数，source 在此是语法需要，实质没有作用。第 10 行 config( ) 方法是需要使用十六进制方式设置背景色，格式是 #007d00。第 18 ～ 20 行的 grid( ) 方法是定义尺度和画布的位置，第 20 行的 columnspan=3 是设置将 3 个字段组成一个字段。此外，本程序在执行时也同时可以在 Python Shell 窗口看到 R、G、B 值的变化。

```
1  # ch19_15.py
2  from tkinter import *
3  def bgUpdate(source):
4      ''' 更改画布背景颜色 '''
5      red = rSlider.get()                              # 读取red值
6      green = gSlider.get()                            # 读取green值
7      blue = bSlider.get( )                            # 读取blue值
8      print("R=%d, G=%d, B=%d" % (red, green, blue))   # 打印色彩数值
9      myColor = "#%02x%02x%02x" % (red, green, blue)   # 将颜色转成十六进制字符串
10     canvas.config(bg=myColor)                        # 设置画布背景颜色
11
12 tk = Tk()
13 canvas = Canvas(tk, width=640, height=240)           # 初始化背景
14 rSlider = Scale(tk, from_=0, to=255, command=bgUpdate)
15 gSlider = Scale(tk, from_=0, to=255, command=bgUpdate)
16 bSlider = Scale(tk, from_=0, to=255, command=bgUpdate)
17 gSlider.set(125)                                     # 设置green是125
18 rSlider.grid(row=0, column=0)
19 gSlider.grid(row=0, column=1)
20 bSlider.grid(row=0, column=2)
21 canvas.grid(row=1, column=0, columnspan=3)
22 mainloop()
```

执行结果



19-3 动画设计

19-3-1 基本动画

动画设计所使用的方法是 move( )，使用格式如下。

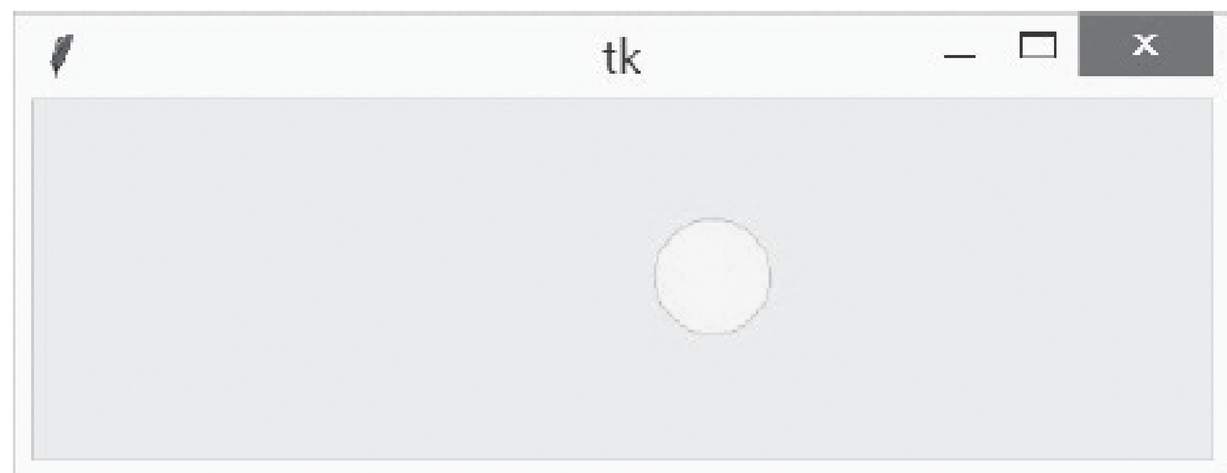
```
canvas.move(ID, xMove, yMove)          # ID 是对象编号
canvas.update( )                       # 强制重绘画布
```

xMove 和 yMove 是 x 和 y 轴的移动距离，单位是像素。

程序实例 ch19_16.py：移动球的设计，每次移动 5 像素。

```
1  # ch19_16.py
2  from tkinter import *
3  import time
4
5  tk = Tk()
6  canvas= Canvas(tk, width=500, height=150)
7  canvas.pack()
8  canvas.create_oval(10,50,60,100,fill='yellow', outline='lightgray')
9  for x in range(0, 80):
10     canvas.move(1, 5, 0)         # ID=1 x轴移动5像素, y轴不变
11     tk.update()                  # 强制tkinter重绘
12     time.sleep(0.05)
```

执行结果



上述第 8 行执行 canvas.create_oval( ) 时，会返回 1，所以第 10 行的 canvas.move( ) 的第一个参数是指第 8 行所建的对象。上述执行时笔者使用循环，第 12 行相当于定义每隔 0.05 秒移动一次。其实只要设置 move( ) 方法的参数就可以往任意方向移动。

程序实例 ch19_17.py：扩大画布高度为 300 像素，每次 x 轴移动 5 像素 , y 轴移动 2 像素。

```
10      canvas.move(1, 5, 2)        # ID=1 x轴移动5像素, y轴移动2像素
```

执行结果 读者可以自行体会球往右下方移动。

上述语句使用 time.sleep(s) 建立时间的延迟，s 表示秒。其实我们也可以使用 canvas.after(s) 建立时间延迟，s 表示千分之一秒，这时可以省略 import time，可以参考 ch19_17_1.py。

程序实例 ch19_17_1.py：重新设计 ch19_17.py。

```
1  # ch19_17_1.py
2  from tkinter import *
3
4  tk = Tk()
5  canvas= Canvas(tk, width=500, height=300)
6  canvas.pack()
7  canvas.create_oval(10,50,60,100,fill='yellow', outline='lightgray')
8  for x in range(0, 80):
9      canvas.move(1, 5, 2)          # ID=1 x轴移动5像素, y轴移动2像素
10     tk.update()                   # 强制tkinter重绘
11     canvas.after(50)
```

执行结果 与 ch19_17.py 相同。

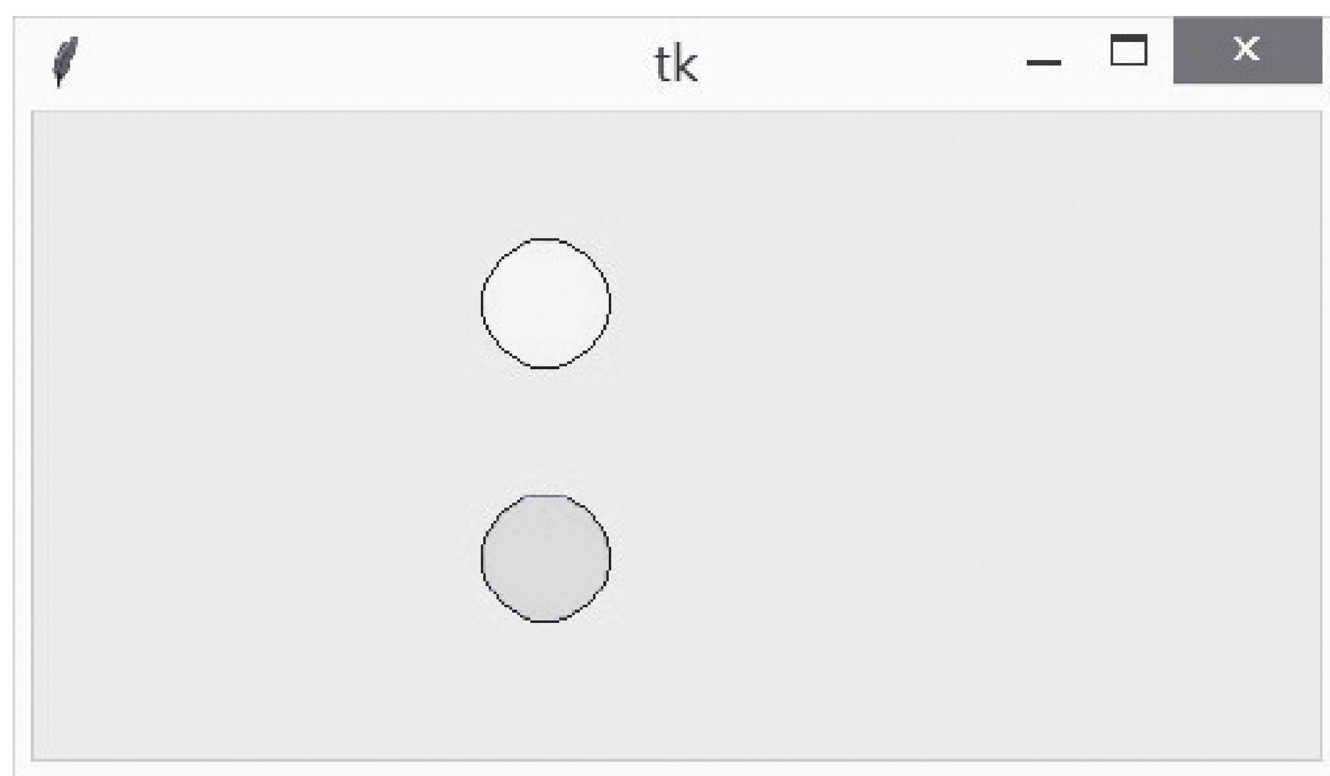
19-3-2　多个球移动的设计

在建立球对象时，可以设置 id 值，未来可以将这个 id 值放入 move( ) 方法内，告知是移动这个球。

程序实例 ch19_18.py：一次移动两个球，第 8 行设置黄色球是 id1，第 9 行设置水蓝色球是 id2。

```
1  # ch19_18.py
2  from tkinter import *
3  import time
4
5  tk = Tk()
6  canvas= Canvas(tk, width=500, height=250)
7  canvas.pack()
8  id1 = canvas.create_oval(10,50,60,100,fill='yellow')
9  id2 = canvas.create_oval(10,150,60,200,fill='aqua')
10 for x in range(0, 80):
11     canvas.move(id1, 5, 0)        # id1 x轴移动5像素, y轴移动0像素
12     canvas.move(id2, 5, 0)        # id2 x轴移动5像素, y轴移动0像素
13     tk.update()                   # 强制tkinter重绘
14     time.sleep(0.05)
```

执行结果



19-3-3　将随机数应用于多个球体的移动

在拉斯维加斯或是澳门赌场，常常可以看到机器赛马的赌局，其实若是将球改成赛马意义是相同的。

1. 赌场可以作弊的方式

假设想让黄色球跑得快一些，赢的概率是 70%，可以利用 randint( ) 产生 1 ～ 100 的随机数，让随机数为 1 ～ 70 时移动黄球，71 ～ 100 时移动水蓝色球。

2. 赌场作弊现形

当我们玩赛马赌局时必须下注，如果赌场要作弊，最佳方式是让下注最少的马匹有较高概率的移动机会，这样钱就滚滚而来了。

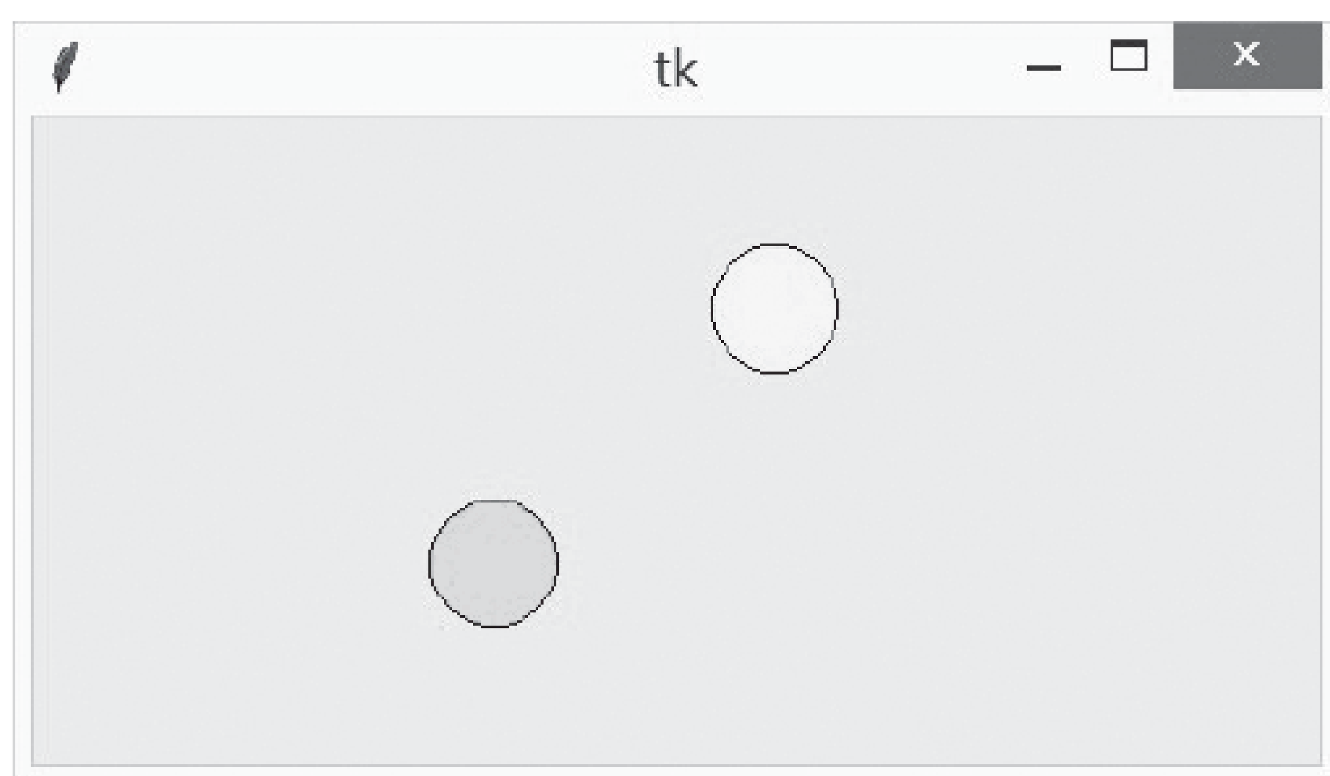
3. 不作弊

可以设计随机数为 1 ～ 50 时移动黄球，51 ～ 100 时移动水蓝色球。

程序实例 ch19_19.py：循环 100 次看哪一个球跑得快，让黄色球每次有 70% 的移动机会。

```
11  for x in range(0, 100):
12      if randint(1,100) > 70:
13          canvas.move(id2, 5, 0)  # id2 x轴移动5像素, y轴移动0像素
14      else:
15          canvas.move(id1, 5, 0)  # id1 x轴移动5像素, y轴移动0像素
16      tk.update()                 # 强制tkinter重绘
17      time.sleep(0.05)
```

执行结果



19-3-4　消息绑定

可以利用系统接收到键盘的消息，做出反应。例如，当按下右移键时，可以控制球往右边移动。假设 Canvas( ) 产生的组件的名称是 canvas，可以如下这样设计函数。

```
def ballMove(event):
    canvas.move(1, 5, 0)                    # 假设移动 5 像素
```

在程序设计函数中对于按下右移键移动球可以如下这样设计。

```
def ballMove(event):
    if event.keysym == 'Right':
        canvas.move(1, 5, 0)
```

对于主程序而言，需使用 canvas.bind_all( ) 函数，执行**消息绑定**工作，它的写法如下。

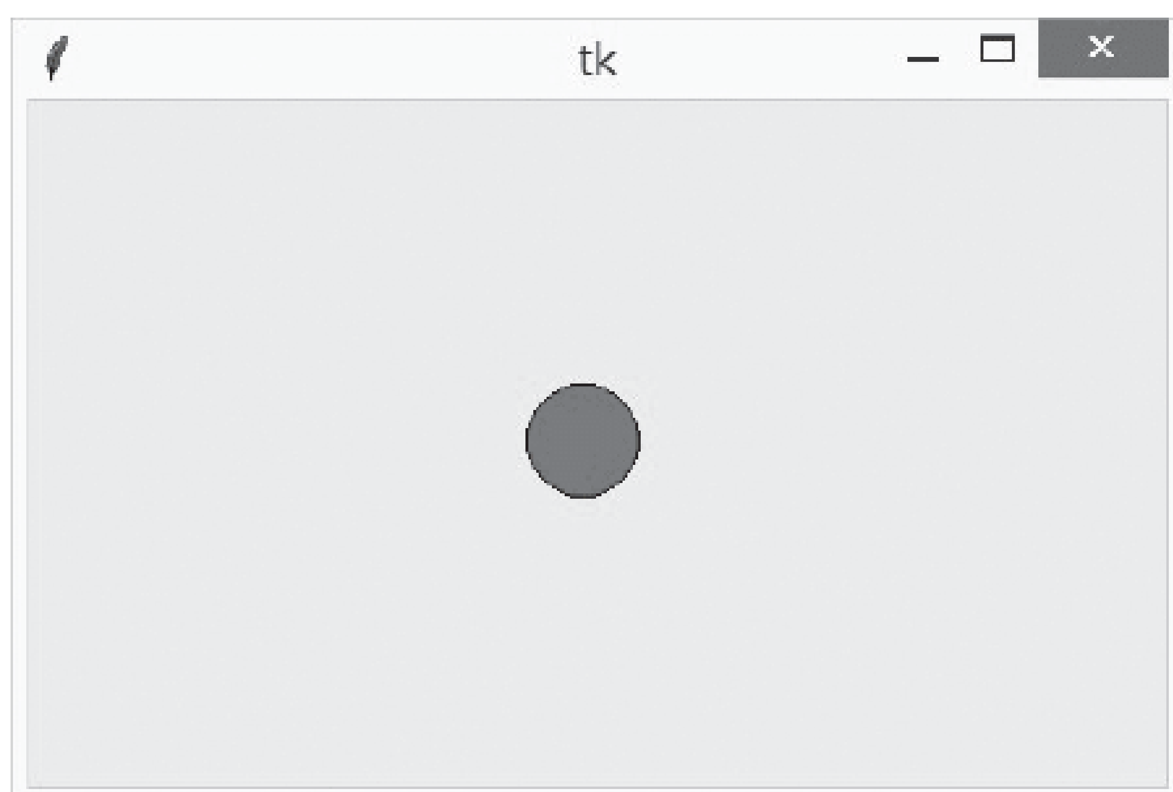
```
canvas.bind_all('<KeyPress-Left>', ballMove)     # 左移键
canvas.bind_all('<KeyPress-Right>', ballMove)    # 右移键
canvas.bind_all('<KeyPress-Up>', ballMove)       # 上移键
canvas.bind_all('<KeyPress-Down>', ballMove)     # 下移键
```

上述函数主要是告知程序所接收到的键盘的消息是什么，然后调用 ballMove( ) 函数执行键盘消息的工作。

程序实例 ch19_20.py：程序开始执行时，在画布中央有一个红球，可以按键盘上的向右、向左、向上、向下键，往右、往左、往上、往下移动球，每次移动 5 像素。

```
1  # ch19_20.py
2  from tkinter import *
3  import time
4  def ballMove(event):
5      if event.keysym == 'Left':  # 左移
6          canvas.move(1, -5, 0)
7      if event.keysym == 'Right': # 右移
8          canvas.move(1, 5, 0)
9      if event.keysym == 'Up':    # 上移
10         canvas.move(1, 0, -5)
11     if event.keysym == 'Down':  # 下移
12         canvas.move(1, 0, 5)
13 tk = Tk()
14 canvas= Canvas(tk, width=500, height=300)
15 canvas.pack()
16 canvas.create_oval(225,125,275,175,fill='red')
17 canvas.bind_all('<KeyPress-Left>', ballMove)
18 canvas.bind_all('<KeyPress-Right>', ballMove)
19 canvas.bind_all('<KeyPress-Up>', ballMove)
20 canvas.bind_all('<KeyPress-Down>', ballMove)
21 mainloop()
```

执行结果



19-3-5 再谈动画设计

19-1 节介绍了 tkinter 的绘图功能，在该节的绘图方法的参数中有说明可以使用 tags 参数将所绘制的对象标上名称，有了这个 tags 名称，未来可以用 canvas.delete（"tags 名称 "）删除此对象，然后可以在新位置再绘制一次此对象，即可以达到对象移动的目的。

注：如果要删除画布内所有对象，可以使用 canvas.delete("all")。

19-3-4 节介绍了键盘的消息绑定，其实也可以使用下面的方式执行鼠标的消息绑定。

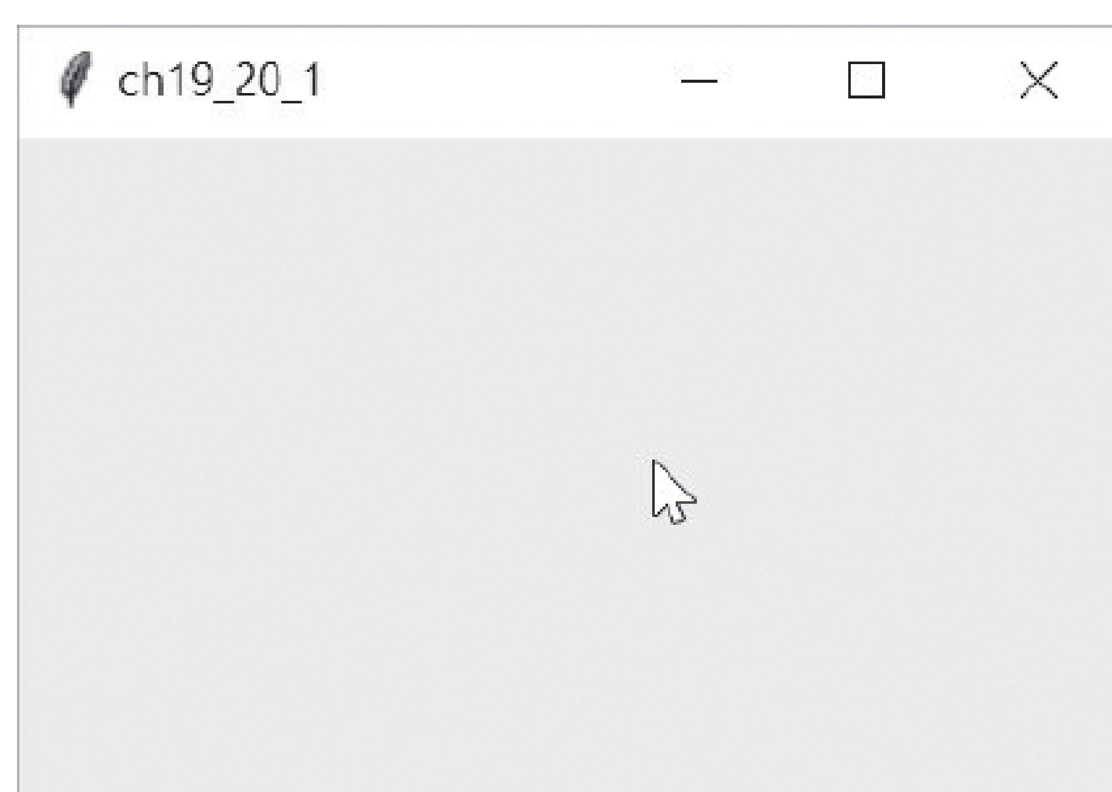
```
canvas.bind('<Button-1>', callback)      # 单击鼠标左键执行 callback 方法
canvas.bind('<Button-2>', callback)      # 单击鼠标中键执行 callback 方法
canvas.bind('<Button-3>', callback)      # 单击鼠标右键执行 callback 方法
canvas.bind('<Motion>', callback)        # 鼠标移动执行 callback 方法
```

上述单击时，鼠标相对组件的位置会被存入事件的 x 和 y 变量。

程序实例 ch19_20_1.py：鼠标事件的基本应用，这个程序在执行时会建立 300×180 的窗口，当单击鼠标左键时，在 Python Shell 窗口中会列出单击时的鼠标坐标。

```
1  # ch19_20_1.py
2  from tkinter import *
3  def callback(event):                        # 事件处理程序
4      print("Clicked at", event.x, event.y)   # 打印坐标
5
6  root = Tk()
7  root.title("ch19_20_1")
8  canvas = Canvas(root,width=300,height=180)
9  canvas.bind("<Button-1>",callback)           # 单击绑定callback
10 canvas.pack()
11
12 root.mainloop()
```

执行结果



下列是 Python Shell 示范输出画面。

```
=================== RESTART: D:/Python/ch19/ch19_20_1.py ===================
Clicked at 159 88
Clicked at 85 60
Clicked at 144 27
```

在程序第 3 行绑定的事件处理程序中必须留意，callback(event) 需有参数 event，event 名称可以自取，这是因为事件会传递事件对象给此事件处理程序。

程序实例 ch19_20_2.py：移动鼠标时可以在窗口右下方看到鼠标目前的坐标。

```
1  # ch19_20_2.py
2  from tkinter import *
3  def mouseMotion(event):              # Mouse移动
4      x = event.x
5      y = event.y
6      textvar = "Mouse location - x:{}, y:{}".format(x,y)
7      var.set(textvar)
8
9  root = Tk()
10 root.title("ch19_20_2")              # 窗口标题
11 root.geometry("300x180")             # 窗口宽300高180
12
13 x, y = 0, 0                          # x,y坐标
14 var = StringVar()
15 text = "Mouse location - x:{}, y:{}".format(x,y)
16 var.set(text)
17
18 lab = Label(root,textvariable=var)   # 建立标签
19 lab.pack(anchor=S,side=RIGHT,padx=10,pady=10)
20
21 root.bind("<Motion>",mouseMotion)    # 增加事件处理程序
22
23 root.mainloop()
```

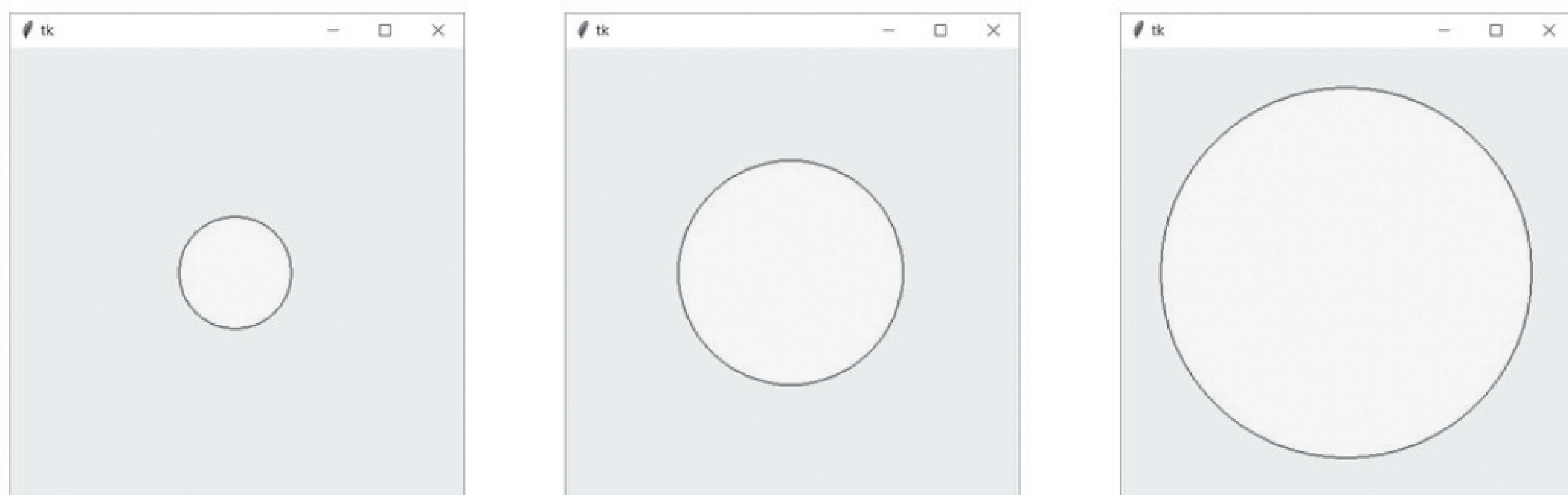
执行结果



程序实例 ch19_20_3.py：单击鼠标左键可以放大圆，单击鼠标右键可以缩小圆。

```
1  # ch19_20_3.py
2  from tkinter import *
3
4  def circleIncrease(event):
5      global r
6      canvas.delete("myCircle")
7      if r < 200:
8          r += 5
9      canvas.create_oval(200-r,200-r,200+r,200+r,fill='yellow',tag="myCircle")
10
11 def circleDecrease(event):
12     global r
13     canvas.delete("myCircle")
14     if r > 5:
15         r -= 5
16     canvas.create_oval(200-r,200-r,200+r,200+r,fill='yellow',tag="myCircle")
17
18 tk = Tk()
19 canvas= Canvas(tk, width=400, height=400)
20 canvas.pack()
21
22 r = 100
23 canvas.create_oval(200-r,200-r,200+r,200+r,fill='yellow',tag="myCircle")
24 canvas.bind('<Button-1>', circleIncrease)
25 canvas.bind('<Button-3>', circleDecrease)
26
27 mainloop()
```

执行结果



19-4 反弹球游戏设计

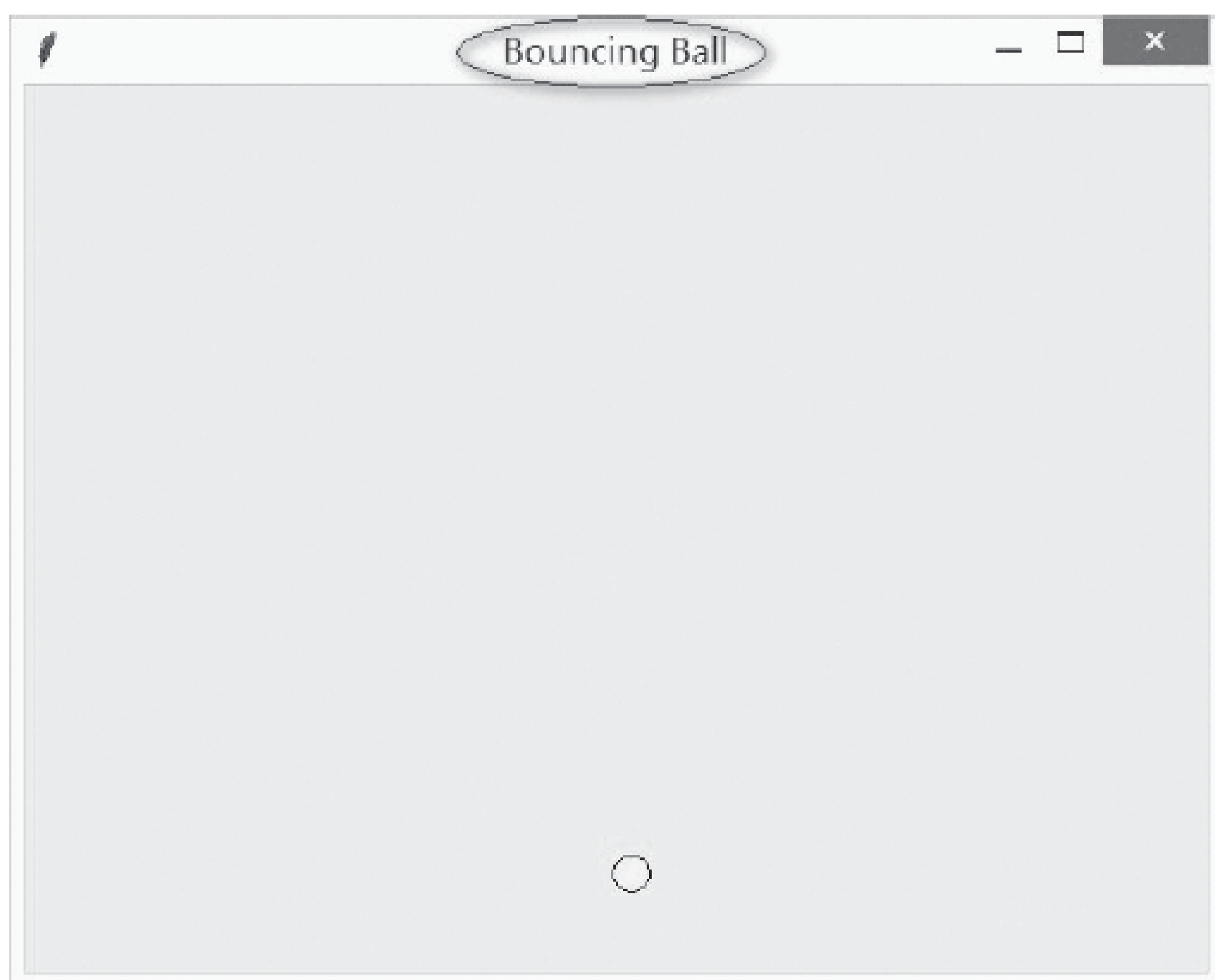
本节将一步一步引导读者设计一个反弹球的游戏。

19-4-1 设计球往下移动

程序实例 ch19_21.py：定义画布窗口名称为 Bouncing Ball，同时定义画布宽度（14 行）与高度（15 行）分别为 640、480。这个球将往下移动然后消失，移到超出画布范围就消失了。

```
1  # ch19_21.py
2  from tkinter import *
3  from random import *
4  import time
5
6  class Ball:
7      def __init__(self, canvas, color, winW, winH):
8          self.canvas = canvas
9          self.id = canvas.create_oval(0, 0, 20, 20, fill=color)  # 建立球对象
10         self.canvas.move(self.id, winW/2, winH/2)   # 设置球最初位置
11     def ballMove(self):
12         self.canvas.move(self.id, 0, step)          # step是正值表示往下移动
13
14 winW = 640                                          # 定义画布宽度
15 winH = 480                                          # 定义画布高度
16 step = 3                                            # 定义速度可想成位移步伐
17 speed = 0.03                                        # 设置移动速度
18
19 tk = Tk()
20 tk.title("Bouncing Ball")                           # 游戏窗口标题
21 tk.wm_attributes('-topmost', 1)                     # 确保游戏窗口在屏幕最上层
22 canvas = Canvas(tk, width=winW, height=winH)
23 canvas.pack()
24 tk.update()
25
26 ball = Ball(canvas, 'yellow', winW, winH)           # 定义球对象
27
28 while True:
29     ball.ballMove()
30     tk.update()
31     time.sleep(speed)                               # 可以控制移动速度
```

执行结果



这个程序由于是一个无限循环（28 ～ 31 行），所以强制关闭画布窗口时，将在 Python Shell 窗口看到错误消息，这无所谓，本章最后实例会改良此情况。整个程序可以用球每次移动的步伐（16 行）和循环第 31 行 time.sleep(speed) 指令的 speed 值，控制球的移动速度。

上述程序笔者建立了 Ball 类，这个类在初始化 __init__( ) 方法中，在第 9 行建立了球对象，第 10 行先设置球是大约在中间位置。另外建立了 ballMove( ) 方法，这个方法会依 step 变量移动，在此例每次往下移动。

19-4-2　设计让球上下反弹

如果想让所设计的球上下反弹，首先需了解 tkinter 模块如何定义对象的位置，其实以这个实例而言，可以使用 coords( ) 方法获得对象位置，它的返回值是对象的左上角和右下角坐标。

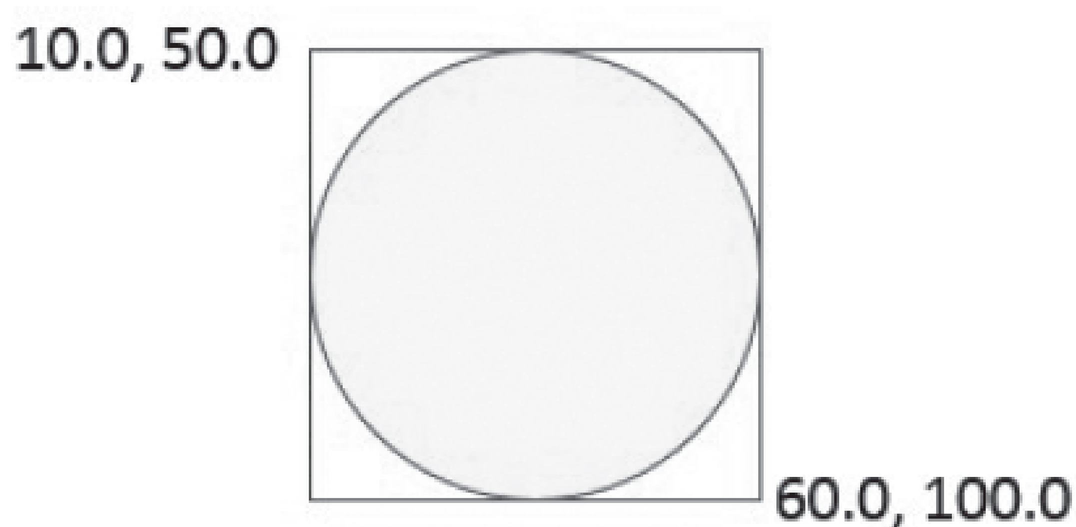
程序实例 ch19_22.py：建立一个球，然后用 coords( ) 方法列出球的位置。

```
1  # ch19_22.py
2  from tkinter import *
3
4  tk = Tk()
5  canvas= Canvas(tk, width=500, height=150)
6  canvas.pack()
7  id = canvas.create_oval(10,50,60,100,fill='yellow', outline='lightgray')
8  ballPos = canvas.coords(id)
9  print(ballPos)
```

执行结果

```
================== RESTART: D:/PythonGUI/ch19/ch19_22.py ==================
[10.0, 50.0, 60.0, 100.0]
>>>
```

上述执行结果可以用以下图示做解说。



相当于可以用 coords( ) 方法获得下列结果。

ballPos[0]：球的左边 x 轴坐标，未来可用于判别是否撞到画布左方。

ballPos[1]：球的上边 y 轴坐标，未来可用于判别是否撞到画布上方。

ballPos[2]：球的右边 x 轴坐标，未来可用于判别是否撞到画布右方。

ballPos[3]：球的下边 y 轴坐标，未来可用于判别是否撞到画布下方。

程序实例 ch19_23.py：改良 ch19_21.py，设计让球可以上下移动，其实这个程序只是更改 Ball 类的内容。

```
 6  class Ball:
 7      def __init__(self, canvas, color, winW, winH):
 8          self.canvas = canvas
 9          self.id = canvas.create_oval(0, 0, 20, 20, fill=color)  # 建立球对象
10          self.canvas.move(self.id, winW/2, winH/2)   # 设置球最初位置
11          self.x = 0                                  # 水平不移动
12          self.y = step                               # 垂直移动单位
13      def ballMove(self):
14          self.canvas.move(self.id, self.x, self.y)   # step是正值表示往下移动
15          ballPos = self.canvas.coords(self.id)
16          if ballPos[1] <= 0:                         # 侦测球是否超过画布上方
17              self.y = step
18          if ballPos[3] >= winH:                      # 侦测球是否超过画布下方
19              self.y = -step
```

执行结果 读者可以观察屏幕，查看球上下移动的结果。

程序第 11 行定义球在 x 轴不移动，第 12 行定义在 y 轴移动单位是 step。第 15 行获得球的位置信息，第 16、17 行侦测如果球撞到画布上方未来球是往下移动 step 单位，第 18、19 行侦测如果球撞到画布下方未来球是往上移动 step 单位（因为是负值）。

19-4-3 设计让球在画布四面反弹

在反弹球游戏中，必须让球在四面皆可反弹，这时需考虑到球在 x 轴的移动，这时原先 Ball 类的 __init__( ) 函数需修改下列两行。

```
11          self.x = 0                          # 水平不移动
12          self.y = step                       # 垂直移动单位
```

下列是更改结果。

```
11          startPos = [-4, -3, -2, -1, 1, 2, 3, 4]    # 球最初x轴位移的随机数
12          shuffle(startPos)                           # 打乱排列
13          self.x = startPos[0]                        # 球最初水平移动单位
14          self.y = step                               # 垂直移动单位
```

上述修改的思路是球局开始时，每个循环 x 轴的移动单位是随机数产生。至于在 ballMove( ) 方法中，需考虑到水平轴的移动可能碰撞画布左边与右边的状况，思路是如果球撞到画布左边，设置球未来在 x 轴的移动是正值，也就是往右移动。

```
18          if ballPos[0] <= 0:                 # 侦测球是否超过画布左方
19              self.x = step
```

如果球撞到画布右边，设置球未来在 x 轴的移动是负值，也就是往左移动。

```
22          if ballPos[2] >= winW:              # 侦测球是否超过画布右方
23              self.x = -step
```

程序实例 ch19_24.py：改良 ch19_23.py 程序，现在球可以在四周移动。

```
 6  class Ball:
 7      def __init__(self, canvas, color, winW, winH):
 8          self.canvas = canvas
 9          self.id = canvas.create_oval(0, 0, 20, 20, fill=color)  # 建立球对象
10          self.canvas.move(self.id, winW/2, winH/2)    # 设置球最初位置
11          startPos = [-4, -3, -2, -1, 1, 2, 3, 4]      # 球最初x轴位移的随机数
12          shuffle(startPos)                            # 打乱排列
13          self.x = startPos[0]                         # 球最初水平移动单位
14          self.y = step                                # 垂直移动单位
15      def ballMove(self):
16          self.canvas.move(self.id, self.x, self.y)    # step是正值表示往下移动
17          ballPos = self.canvas.coords(self.id)
18          if ballPos[0] <= 0:                          # 侦测球是否超过画布左方
19              self.x = step
20          if ballPos[1] <= 0:                          # 侦测球是否超过画布上方
21              self.y = step
22          if ballPos[2] >= winW:                       # 侦测球是否超过画布右方
23              self.x = -step
24          if ballPos[3] >= winH:                       # 侦测球是否超过画布下方
25              self.y = -step
```

执行结果 读者可以观察屏幕，查看球在画布四周移动的结果。

19-4-4　建立球拍

首先建立一个静止的球拍，此时可以建立 Racket 类，在这个类中设置了它的初始大小与位置。

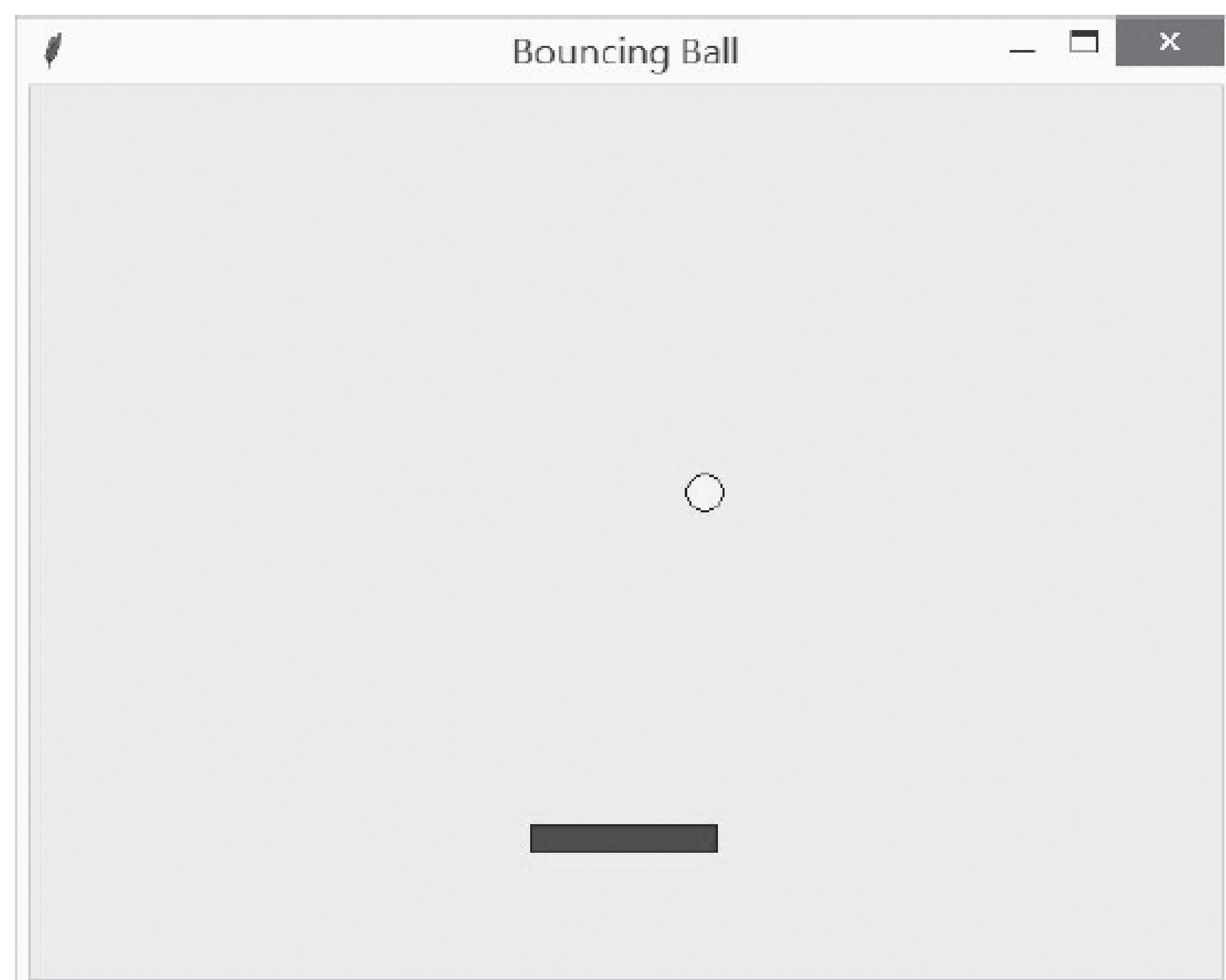
程序实例 ch19_25.py：扩充 ch19_24.py，主要是增加球拍设计，在这里先增加球拍类。在这个类中，在第 29 行设计了球拍的大小和颜色，第 30 行设置了最初球拍的位置。

```
26  class Racket:
27      def __init__(self, canvas, color):
28          self.canvas = canvas
29          self.id = canvas.create_rectangle(0,0,100,15, fill=color)   # 球拍对象
30          self.canvas.move(self.id, 270, 400)                         # 球拍位置
```

另外，在主程序中增加了建立一个球拍对象。

```
44  racket = Racket(canvas, 'purple')              # 定义紫色球拍
```

执行结果



19-4-5 设计球拍移动

由于是假设使用键盘的右移和左移键移动球拍，所以可以在 Ractet 的 __init__( ) 函数内增加，使用 bind_all( ) 方法绑定键盘按键发生时的移动方式。

```
32         self.canvas.bind_all('<KeyPress-Right>', self.moveRight)    # 绑定按往右键
33         self.canvas.bind_all('<KeyPress-Left>', self.moveLeft)      # 绑定按往左键
```

所以在 Ractet 类内增加下列 moveRight( ) 和 moveLeft( ) 的设计。

```
41     def moveLeft(self, event):                      # 球拍每次向左移动的单位数
42         self.x = -3
43     def moveRight(self, event):                     # 球拍每次向右移动的单位数
44         self.x = 3
```

上述设计相当于每次的位移量是 3，如果游戏设有等级，可以让新手位移量增加，随等级增加让位移量减少。此外，这个程序增加了球拍移动主体设计。

```
34     def racketMove(self):                           # 设计球拍移动
35         self.canvas.move(self.id, self.x, 0)
36         pos = self.canvas.coords(self.id)
37         if pos[0] <= 0:                             # 移动时是否碰到画布左边
38             self.x = 0
39         elif pos[2] >= winW:                        # 移动时是否碰到画布右边
40             self.x = 0
```

主程序也将新增球拍移动调用。

```
61 while True:
62     ball.ballMove()
63     racket.racketMove()
64     tk.update()
65     time.sleep(speed)                               # 可以控制移动速度
```

程序实例 ch19_26.py：扩充 ch19_25.py 的功能，增加设计让球拍左右可以移动，下列程序第 31 行是设置程序开始时，球拍位移是 0，下列是球拍类的内容。

```
26 class Racket:
27     def __init__(self, canvas, color):
28         self.canvas = canvas
29         self.id = canvas.create_rectangle(0,0,100,15, fill=color)  # 球拍对象
30         self.canvas.move(self.id, 270, 400)                        # 球拍位置
31         self.x = 0
32         self.canvas.bind_all('<KeyPress-Right>', self.moveRight)   # 绑定按往右键
33         self.canvas.bind_all('<KeyPress-Left>', self.moveLeft)     # 绑定按往左键
34     def racketMove(self):                        # 设计球拍移动
35         self.canvas.move(self.id, self.x, 0)
36         pos = self.canvas.coords(self.id)
37         if pos[0] <= 0:                          # 移动时是否碰到画布左边
38             self.x = 0
39         elif pos[2] >= winW:                     # 移动时是否碰到画布右边
40             self.x = 0
41     def moveLeft(self, event):                   # 球拍每次向左移动的单位数
42         self.x = -3
43     def moveRight(self, event):                  # 球拍每次向右移动的单位数
44         self.x = 3
```

下列是主程序内容。

```
58 racket = Racket(canvas, 'purple')              # 定义紫色球拍
59 ball = Ball(canvas, 'yellow', winW, winH)      # 定义球对象
60
61 while True:
62     ball.ballMove()
63     racket.racketMove()
64     tk.update()
65     time.sleep(speed)                          # 可以控制移动速度
```

执行结果 读者可以观察屏幕，球拍已经可以左右移动了。

19-4-6　球拍与球碰撞的处理

在上述程序的执行结果中，球碰到球拍基本上是可以穿透过去，这一节将讲解碰撞的处理，首先可以增加将 Racket 类传给 Ball 类，如下所示。

```
6  class Ball:
7      def __init__(self, canvas, color, winW, winH, racket):
8          self.canvas = canvas
9          self.racket = racket
```

当然在主程序建立 Ball 类对象时需修改调用如下。

```
67  racket = Racket(canvas, 'purple')              # 定义紫色球拍
68  ball = Ball(canvas,'yellow',winW,winH,racket)  # 定义球对象
```

在 Ball 类中需增加球是否碰到球拍的方法，如果碰到就让球路径往上反弹。

```
33          if self.hitRacket(ballPos) == True:       # 侦测是否撞到球拍
34              self.y = -step
```

在 Ball 类 ballMove( ) 方法上方需增加下列 hitRacket( ) 方法，检测球是否碰撞球拍，如果碰撞了会返回 True，否则返回 False。

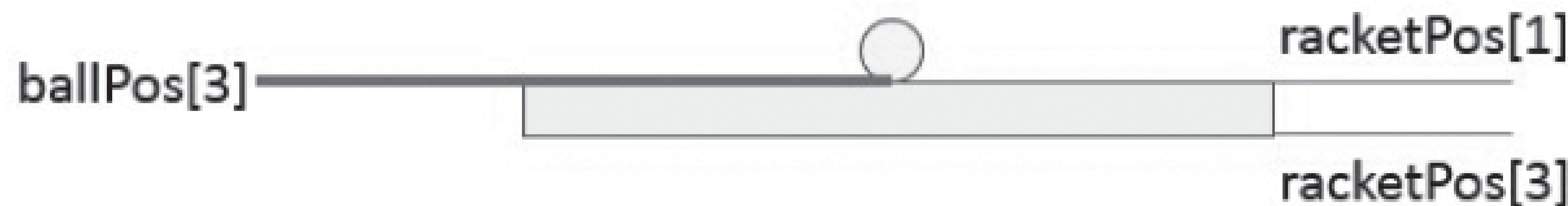
```
16      def hitRacket(self, ballPos):
17          racketPos = self.canvas.coords(self.racket.id)
18          if ballPos[2] >= racketPos[0] and ballPos[0] <= racketPos[2]:
19              if ballPos[3] >= racketPos[1] and ballPos[3] <= racketPos[3]:
20                  return True
21          return False
```

上述侦测球是否撞到球拍必须符合以下两个条件。

（1）球的右侧 x 轴坐标 ballPos[2] 大于球拍左侧 x 坐标 racketPos[0]，同时球的左侧 x 坐标 ballPos[0] 小于球拍右侧 x 坐标 racketPos[2]。



（2）球的下方 y 坐标 ballPos[3] 大于球拍上方的 y 坐标 racketPos[1]，同时必须小于球拍下方的 y 坐标 reaketPos[3]。读者可能奇怪为何不是侦测碰到球拍上方即可，主要是球不是一次移动 1 像素，如果移动 3 像素，很可能会跳过球拍上方。



下列是球的可能移动方式。



程序实例 ch19_27.py：扩充 ch19_26.py，当球碰撞到球拍时会反弹，下列是完整的 Ball 类的设计。

```
6  class Ball:
7      def __init__(self, canvas, color, winW, winH, racket):
8          self.canvas = canvas
9          self.racket = racket
10         self.id = canvas.create_oval(0, 0, 20, 20, fill=color)  # 建立球对象
11         self.canvas.move(self.id, winW/2, winH/2)   # 设置球最初位置
12         startPos = [-4, -3, -2, -1, 1, 2, 3, 4]     # 球最初x轴位移的随机数
13         shuffle(startPos)                           # 打乱排列
14         self.x = startPos[0]                        # 球最初水平移动单位
15         self.y = step                               # 垂直移动单位
16     def hitRacket(self, ballPos):
17         racketPos = self.canvas.coords(self.racket.id)
18         if ballPos[2] >= racketPos[0] and ballPos[0] <= racketPos[2]:
19             if ballPos[3] >= racketPos[1] and ballPos[3] <= racketPos[3]:
20                 return True
21         return False
22     def ballMove(self):
23         self.canvas.move(self.id, self.x, self.y)   # step是正值表示往下移动
24         ballPos = self.canvas.coords(self.id)
25         if ballPos[0] <= 0:                         # 侦测球是否超过画布左方
26             self.x = step
27         if ballPos[1] <= 0:                         # 侦测球是否超过画布上方
28             self.y = step
29         if ballPos[2] >= winW:                      # 侦测球是否超过画布右方
30             self.x = -step
31         if ballPos[3] >= winH:                      # 侦测球是否超过画布下方
32             self.y = -step
33         if self.hitRacket(ballPos) == True:         # 侦测是否撞到球拍
34             self.y = -step
```

执行结果 读者可以观察屏幕，球碰撞到球拍时会反弹。

19-4-7 完整的游戏

在游戏中，若是球碰触画布底端应该让游戏结束，此时首先在第 16 行 Ball 类的 __init__( ) 函数中声明 notTouchBottom 为 True，为了让玩家可以缓冲，笔者此时也设置球局开始时球是往上移动（第 15 行），如下所示。

```
15         self.y = -step                              # 球先往上垂直移动单位
16         self.notTouchBottom = True                  # 未接触画布底端
```

我们修改主程序中的循环如下。

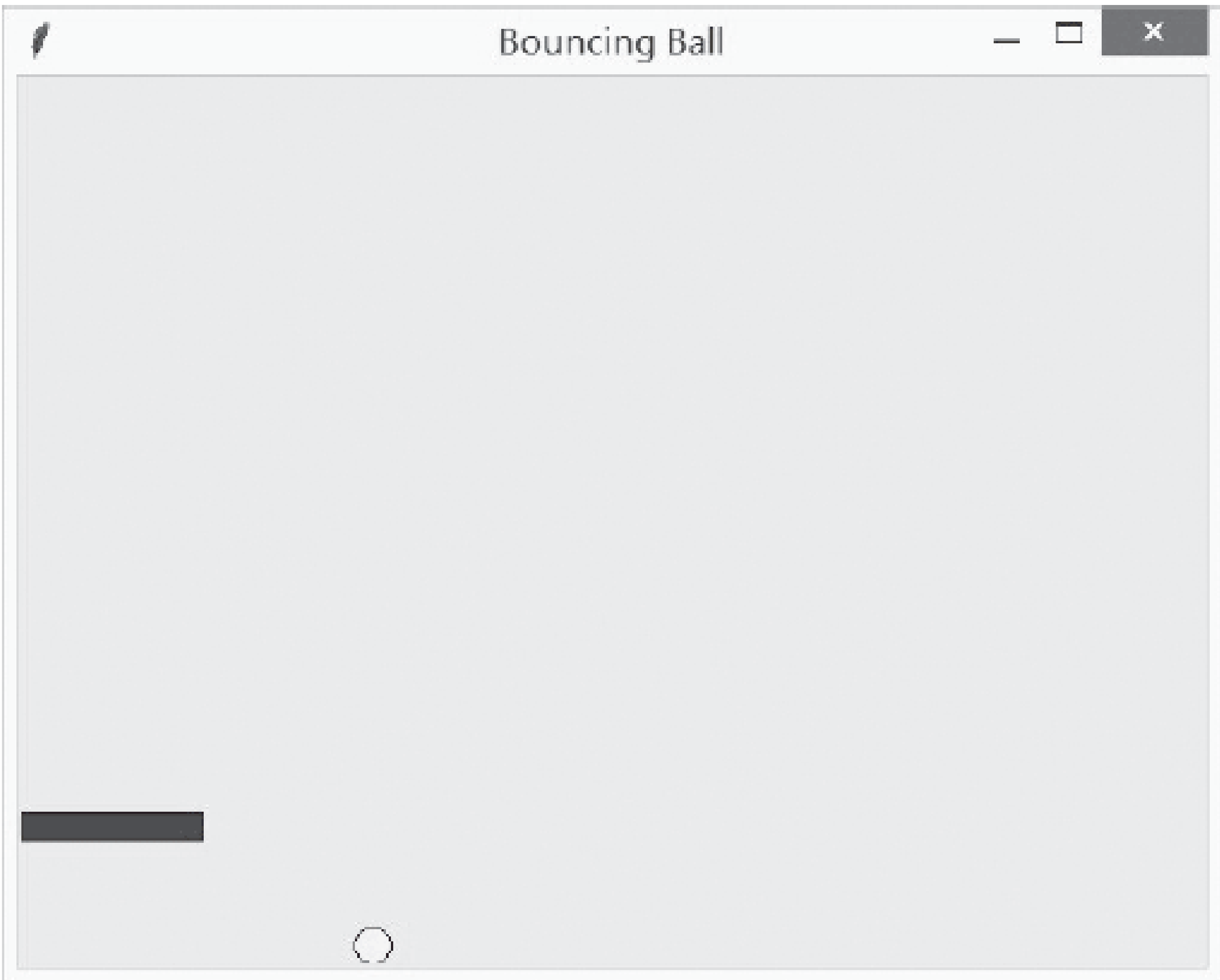
```
73 while ball.notTouchBottom:                          # 如果球未接触画布底端
74     try:
75         ball.ballMove()
76     except:
77         print("单击关闭按钮终止程序执行")
78         break
79     racket.racketMove()
80     tk.update()
81     time.sleep(speed)                               # 可以控制移动速度
```

最后在 Ball 类的 ballMove( ) 方法中侦测球是否接触画布底端，如果是则将 notTouchBottom 设为 False，这个 False 将让主程序的循环停止执行。同时捕捉异常时如果单击 Bouncing Ball 窗口的“关闭”按钮，这样就不会再有错误消息产生了。

程序实例 ch19_28.py：完整的反弹球设计。

```
1  # ch19_28.py
2  from tkinter import *
3  from random import *
4  import time
5
6  class Ball:
7      def __init__(self, canvas, color, winW, winH, racket):
8          self.canvas = canvas
9          self.racket = racket
10         self.id = canvas.create_oval(0, 0, 20, 20, fill=color)  # 建立球对象
11         self.canvas.move(self.id, winW/2, winH/2)    # 设置球最初位置
12         startPos = [-4, -3, -2, -1, 1, 2, 3, 4]      # 球最初x轴位移的随机数
13         shuffle(startPos)                            # 打乱排列
14         self.x = startPos[0]                         # 球最初水平移动单位
15         self.y = -step                               # 球先往上垂直移动单位
16         self.notTouchBottom = True                   # 未接触画布底端
17     def hitRacket(self, ballPos):
18         racketPos = self.canvas.coords(self.racket.id)
19         if ballPos[2] >= racketPos[0] and ballPos[0] <= racketPos[2]:
20             if ballPos[3] >= racketPos[1] and ballPos[3] <= racketPos[3]:
21                 return True
22         return False
23     def ballMove(self):
24         self.canvas.move(self.id, self.x, self.y)    # step是正值表示往下移动
25         ballPos = self.canvas.coords(self.id)
26         if ballPos[0] <= 0:                          # 侦测球是否超过画布左方
27             self.x = step
28         if ballPos[1] <= 0:                          # 侦测球是否超过画布上方
29             self.y = step
30         if ballPos[2] >= winW:                       # 侦测球是否超过画布右方
31             self.x = -step
32         if ballPos[3] >= winH:                       # 侦测球是否超过画布下方
33             self.y = -step
34         if self.hitRacket(ballPos) == True:          # 侦测是否撞到球拍
35             self.y = -step
36         if ballPos[3] >= winH:                       # 如果球接触到画布底端
37             self.notTouchBottom = False
38 class Racket:
39     def __init__(self, canvas, color):
40         self.canvas = canvas
41         self.id = canvas.create_rectangle(0,0,100,15, fill=color)   # 球拍对象
42         self.canvas.move(self.id, 270, 400)                         # 球拍位置
43         self.x = 0
44         self.canvas.bind_all('<KeyPress-Right>', self.moveRight)    # 绑定按往右键
45         self.canvas.bind_all('<KeyPress-Left>', self.moveLeft)      # 绑定按往左键
46     def racketMove(self):                            # 设计球拍移动
47         self.canvas.move(self.id, self.x, 0)
48         racketPos = self.canvas.coords(self.id)
49         if racketPos[0] <= 0:                        # 移动时是否碰到画布左边
50             self.x = 0
51         elif racketPos[2] >= winW:                   # 移动时是否碰到画布右边
52             self.x = 0
53     def moveLeft(self, event):                       # 球拍每次向左移动的单位数
54         self.x = -3
55     def moveRight(self, event):                      # 球拍每次向右移动的单位数
56         self.x = 3
57
58 winW = 640                                           # 定义画布宽度
59 winH = 480                                           # 定义画布高度
60 step = 3                                             # 定义速度可想成位移步伐
61 speed = 0.01                                         # 设置移动速度
62
63 tk = Tk()
64 tk.title("Bouncing Ball")                            # 游戏窗口标题
65 tk.wm_attributes('-topmost', 1)                      # 确保游戏窗口在屏幕最上层
66 canvas = Canvas(tk, width=winW, height=winH)
67 canvas.pack()
68 tk.update()
69
70 racket = Racket(canvas, 'purple')                    # 定义紫色球拍
71 ball = Ball(canvas,'yellow',winW,winH,racket)        # 定义球对象
72
73 while ball.notTouchBottom:                           # 如果球未接触画布底端
74     try:
75         ball.ballMove()
76     except:
77         print("单击关闭按钮终止程序执行")
78         break
79     racket.racketMove()
80     tk.update()
81     time.sleep(speed)                                # 可以控制移动速度
```

执行结果



19-5 专题——使用 tkinter 处理谢尔宾斯基三角形

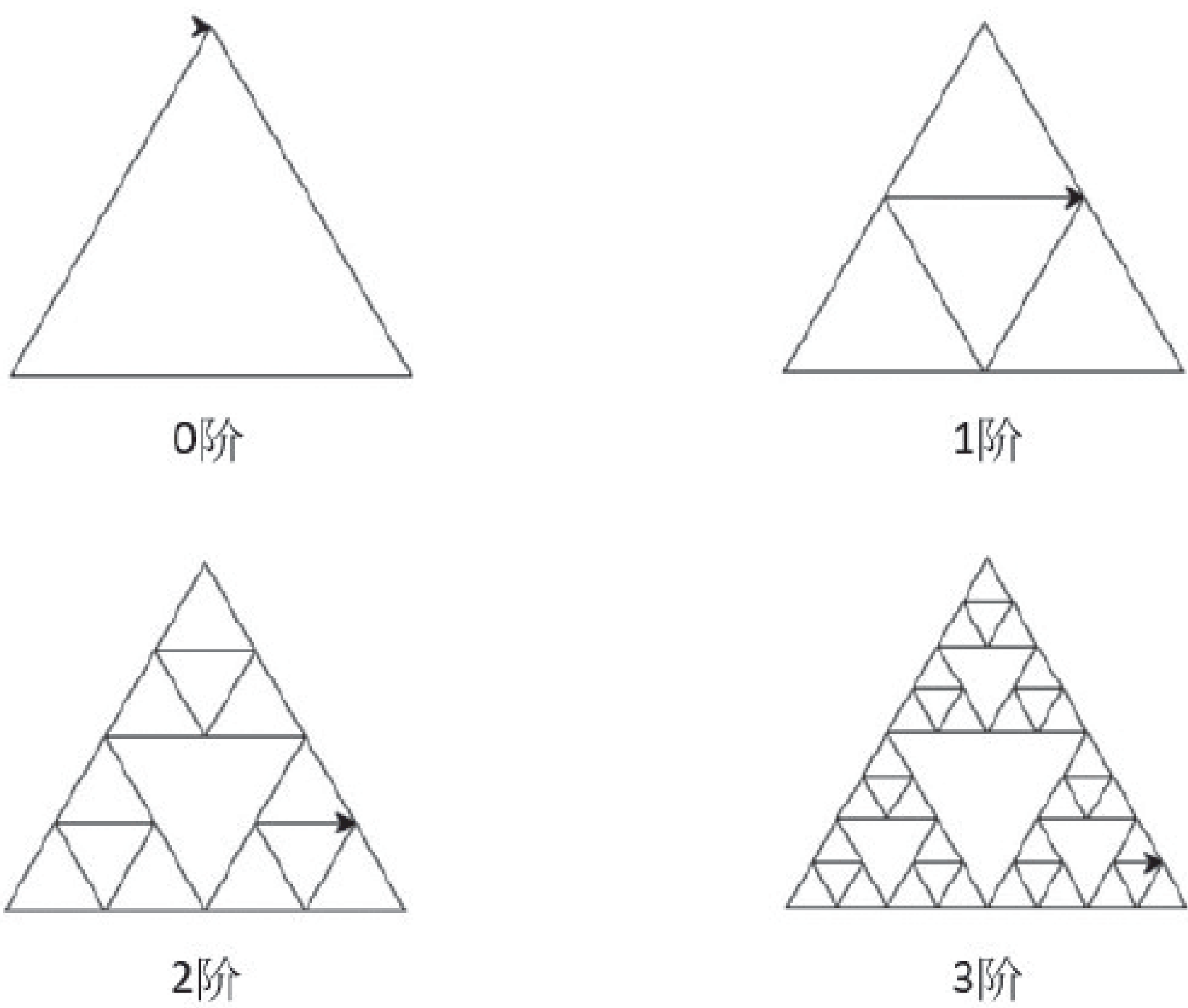
谢尔宾斯基三角形（Sierpinski Triangle）是由波兰数学家谢尔宾斯基在 1915 年提出的一种三角形概念，这个三角形本质上是碎形（Fractal）。所谓碎形是一个几何图形，它可以分为许多部分，每个部分都是整体的缩小版。这个三角形建立的步骤如下。

（1）建立一个等边三角形，这个三角形称为 0 阶（order = 0）谢尔宾斯基三角形。

（2）将三角形各边中点连接，称为 1 阶谢尔宾斯基三角形。

（3）中间三角形不变，将其他 3 个三角形各边中点连接，称为 2 阶谢尔宾斯基三角形。

（4）使用 11-6 节递归式函数概念，重复上述步骤，即可产生 3 阶、4 阶或更高阶的谢尔宾斯基三角形。



使用 tkinter 解这个题目最大的优点是可以在 GUI 接口随时更改阶乘数字，然后可以在画布显示执行结果。

在这一节计划介绍另一个组件（widget）框架 Frame，也可将此想象成是容器组件，这个框架 Frame 通常用于碰上复杂的 GUI 设计时，可以将部分其他 tkinter 组件组织在此框架内（可想成是容器），如此可以简化 GUI。它的建构方法如下。

Frame（父对象，options，…）

Frame( ) 方法的第一个参数是**父对象**，表示这个框架将建立在哪一个父对象内。下面是 Frame( ) 方法内其他常用的 options 参数。

bg 或 background：背景色彩。

borderwidth 或 bd：标签边界宽度，默认是 2。

cursor：当鼠标光标在框架上时的光标外形。

height：框架的高度，单位是像素。

highlightbackground：当框架没有取得焦点时的颜色。

highlightcolor：当框架取得焦点时的颜色。

highlighthickness：当框架取得焦点时的厚度。

relief：默认是 relief=FLAT，可由此控制框架外框。

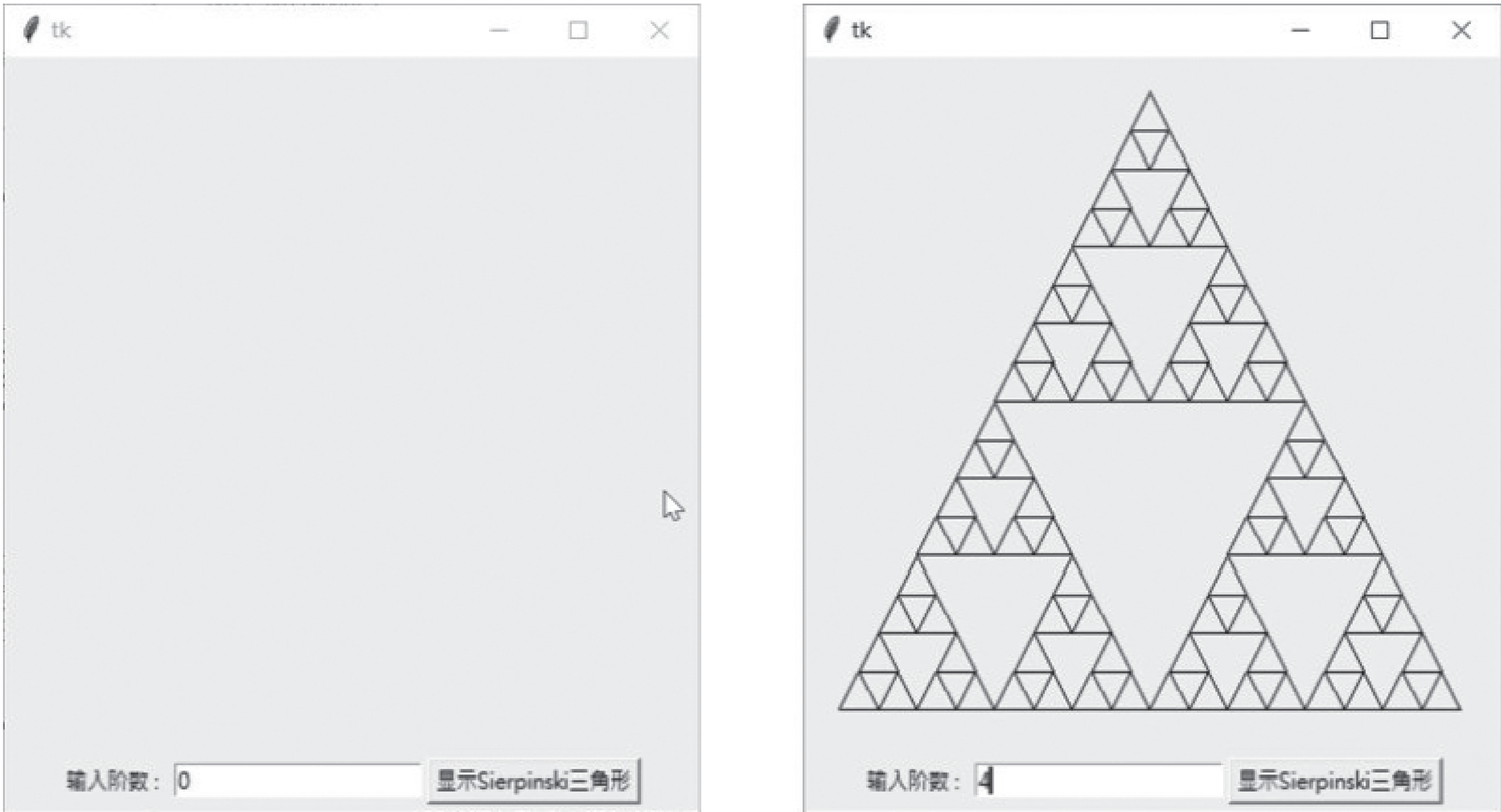
width：框架的宽度，单位是像素，省略时会自行调整为实际宽度。

程序实例 ch19_29.py：设计谢尔宾斯基三角形（Sierpinski Triangle），这个程序基本过程是在 tk 窗口内分别建立 Canvas( ) 对象 canvas 和 Frame( ) 对象 frame，然后在 canvas 对象内绘制谢尔宾斯基三角形。在 frame 对象内建立标签 Label、文本框 Entry 和按钮 Button，这是用于建立输入绘制谢尔宾斯基三角形的阶乘数与正式控制执行。

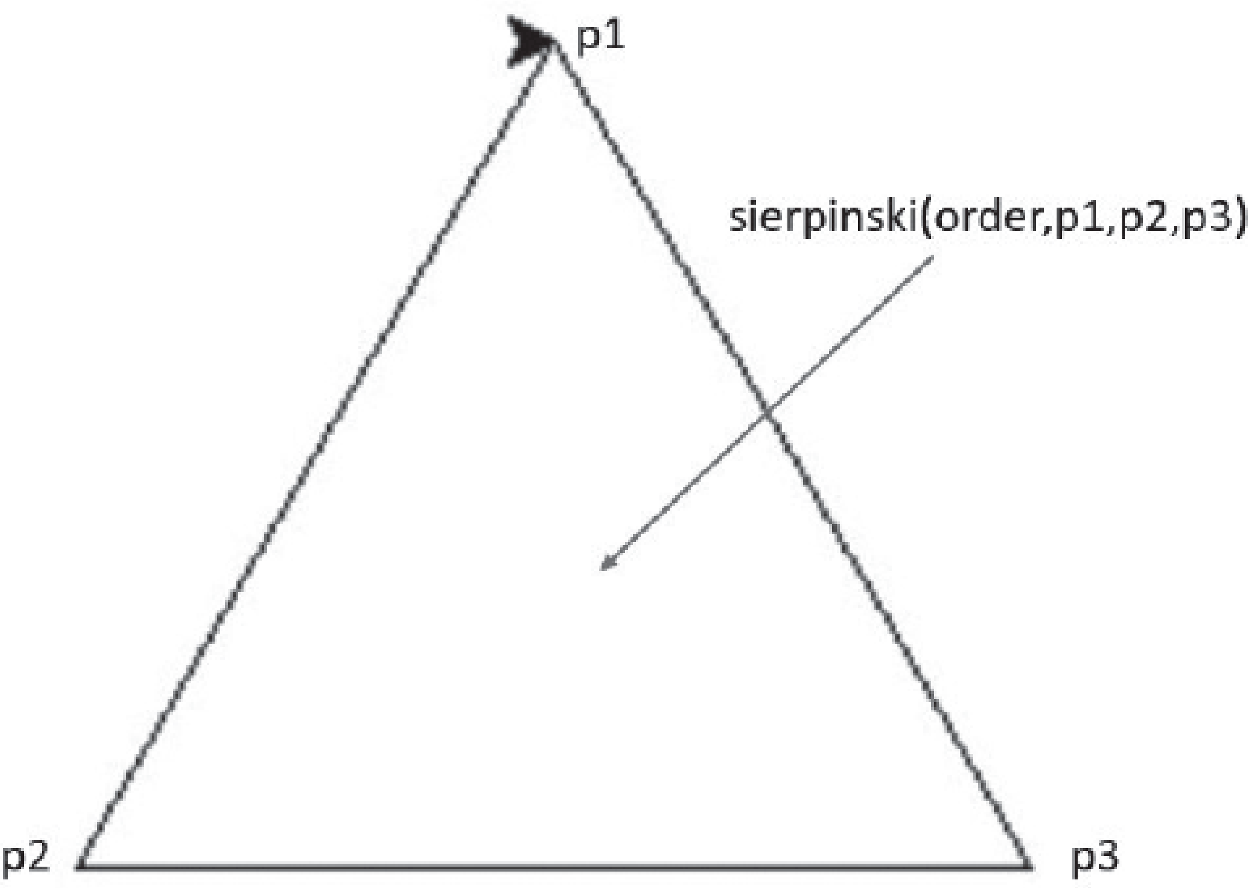
```
1  # ch19_29.py
2  from tkinter import *
3  # 依据特定阶级数绘制Sierpinski三角形
4  def sierpinski(order, p1, p2, p3):
5      if order == 0:        # 阶级数为0
6          # 将3个点连接绘制成三角形
7          drawLine(p1, p2)
8          drawLine(p2, p3)
9          drawLine(p3, p1)
10     else:
11         # 取得三角形各边长的中点
12         p12 = midpoint(p1, p2)
13         p23 = midpoint(p2, p3)
14         p31 = midpoint(p3, p1)
15         # 递归调用处理绘制三角形
16         sierpinski(order - 1, p1, p12, p31)
17         sierpinski(order - 1, p12, p2, p23)
18         sierpinski(order - 1, p31, p23, p3)
19 # 绘制p1和p2之间的线条
20 def drawLine(p1,p2):
21     canvas.create_line(p1[0],p1[1],p2[0],p2[1],tags="myline")
22 # 返回两点的中间值
23 def midpoint(p1, p2):
24     p = [0,0]                            # 初值设置
25     p[0] = (p1[0] + p2[0]) / 2
26     p[1] = (p1[1] + p2[1]) / 2
27     return p
28 # 显示
29 def show():
30     canvas.delete("myline")
31     p1 = [200, 20]
32     p2 = [20, 380]
33     p3 = [380,380]
34     sierpinski(order.get(), p1, p2, p3)
35
36 # main
37 tk = Tk()
38 canvas = Canvas(tk, width=400, height=400)      # 建立画布
39 canvas.pack()
40
```

```
41  frame = Frame(tk)                              # 建立框架
42  frame.pack(padx=5, pady=5)
43  # 在框架Frame内建立标签Label, 输入阶乘数Entry, 按钮Button
44  Label(frame, text="输入阶数 : ").pack(side=LEFT)
45  order = IntVar()
46  order.set(0)
47  entry = Entry(frame, textvariable=order).pack(side=LEFT,padx=3)
48  Button(frame, text="显示Sierpinski三角形",
49         command=show).pack(side=LEFT)
50
51  tk.mainloop()
```

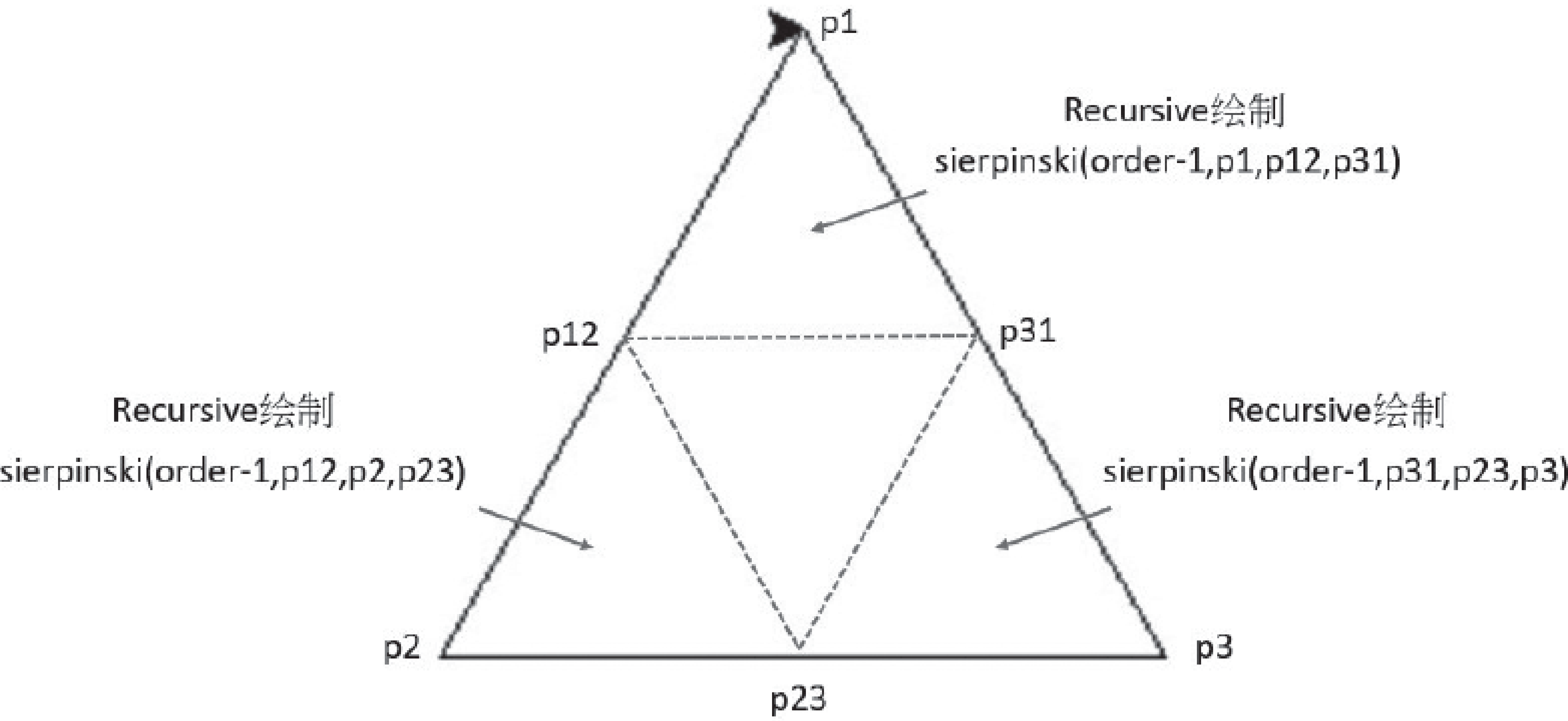
执行结果



上述程序绘制的第一个 0 阶谢尔宾斯基三角形如下。

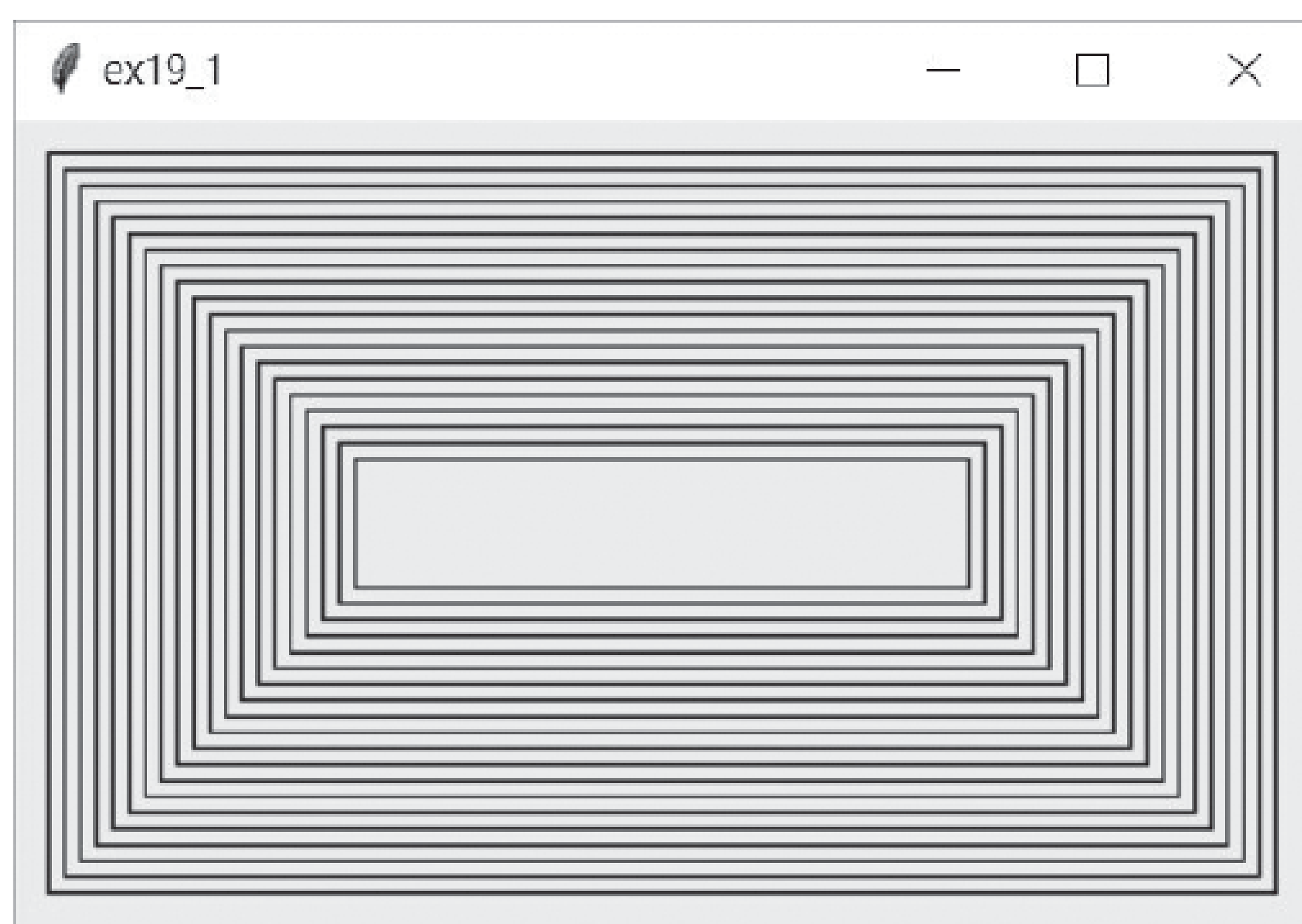


递归调用绘制谢尔宾斯基三角形如下。

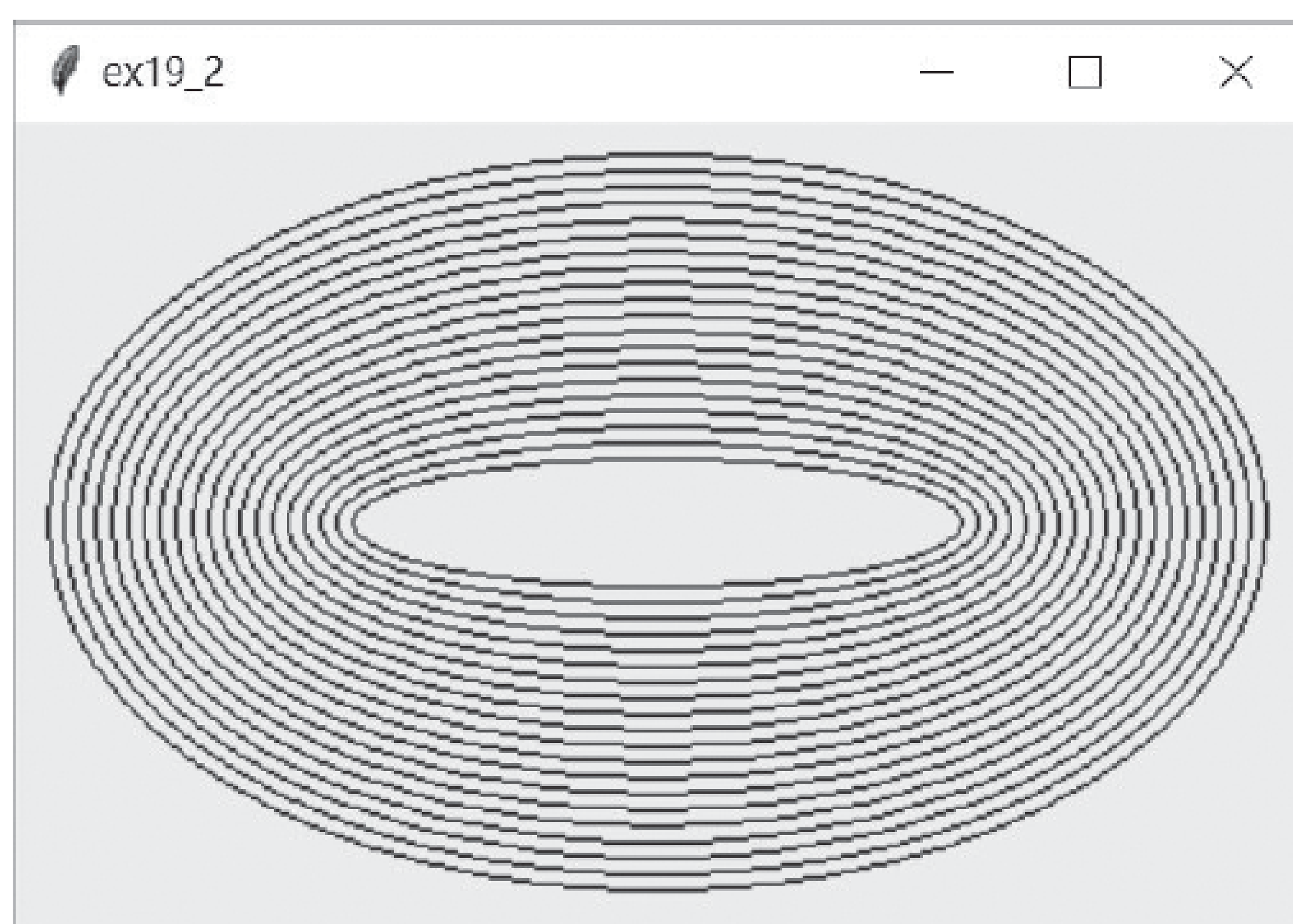


习题

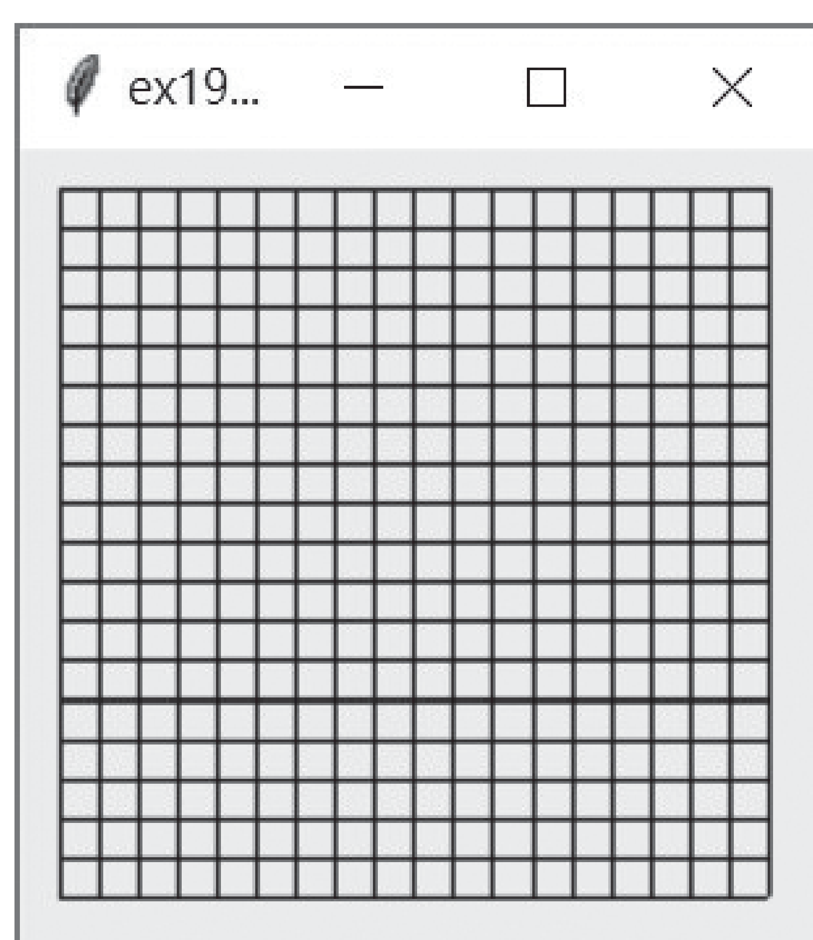
1. 写一个程序，画布大小是 400×250，由外往内绘制，每次宽和高减 10，可以显示 20 个矩形。(19-1 节)



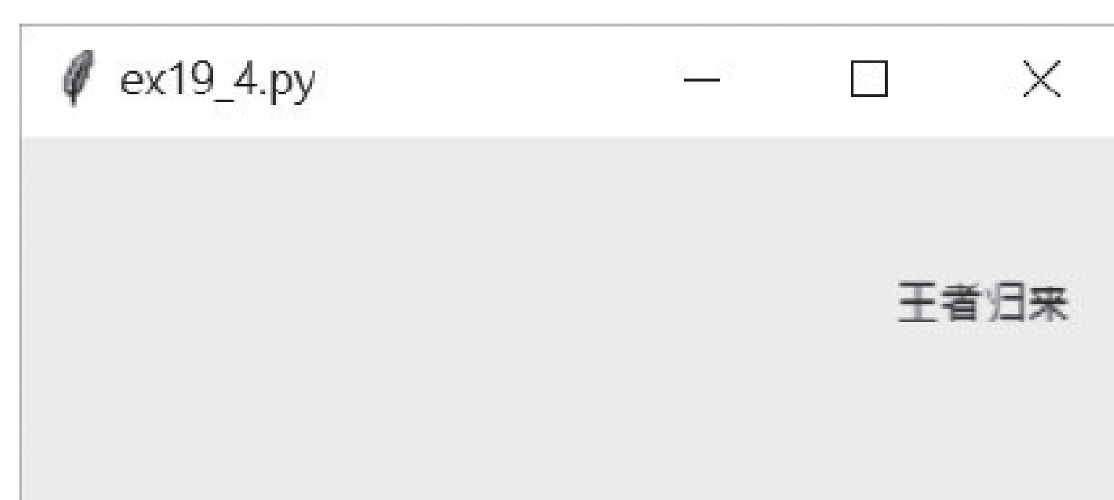
2. 写一个程序，画布大小是 400×250，由外往内绘制椭圆，每次椭圆宽和高减 10，可以显示 20 个椭圆。(19-1 节)



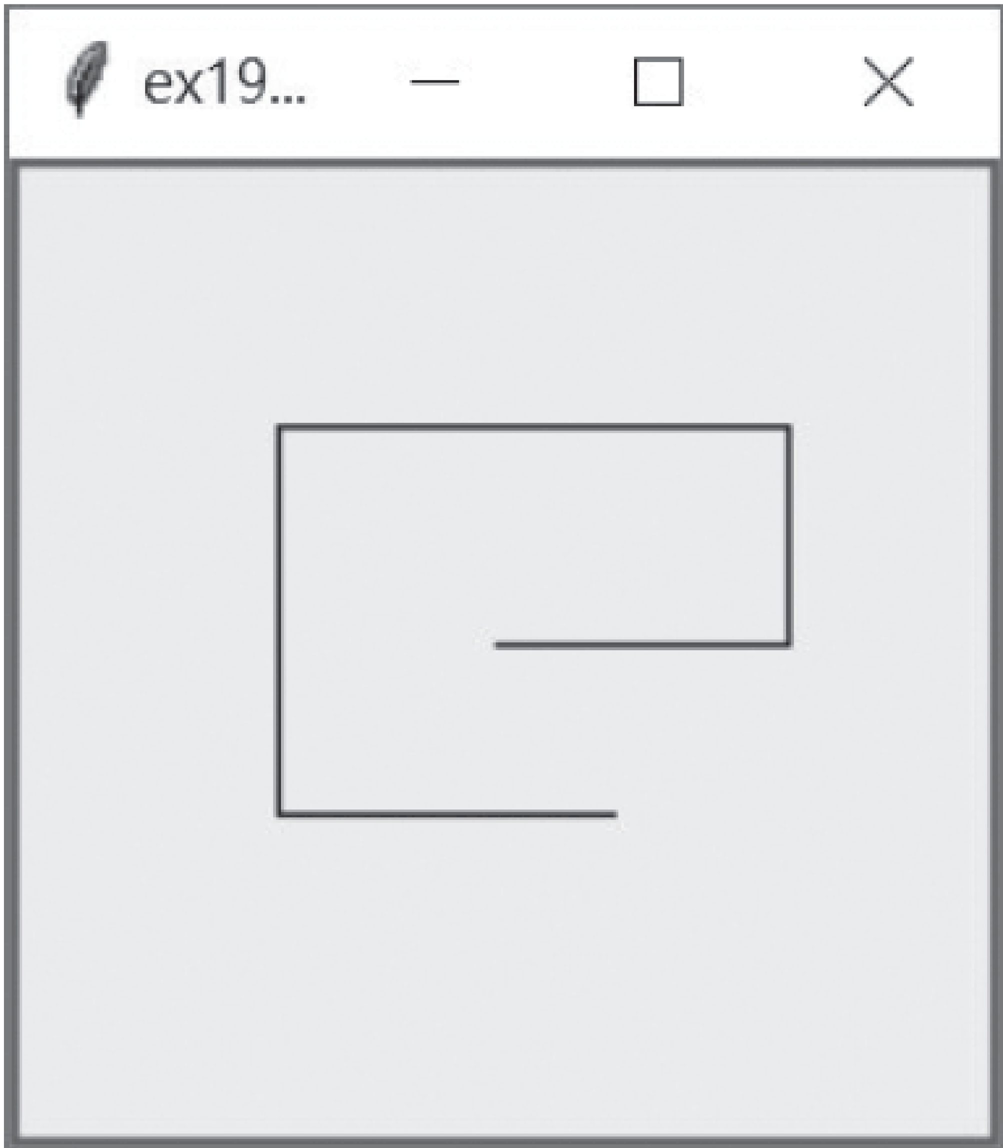
3. 写一个程序，可以显示 15×15 的网格。(19-1 节)



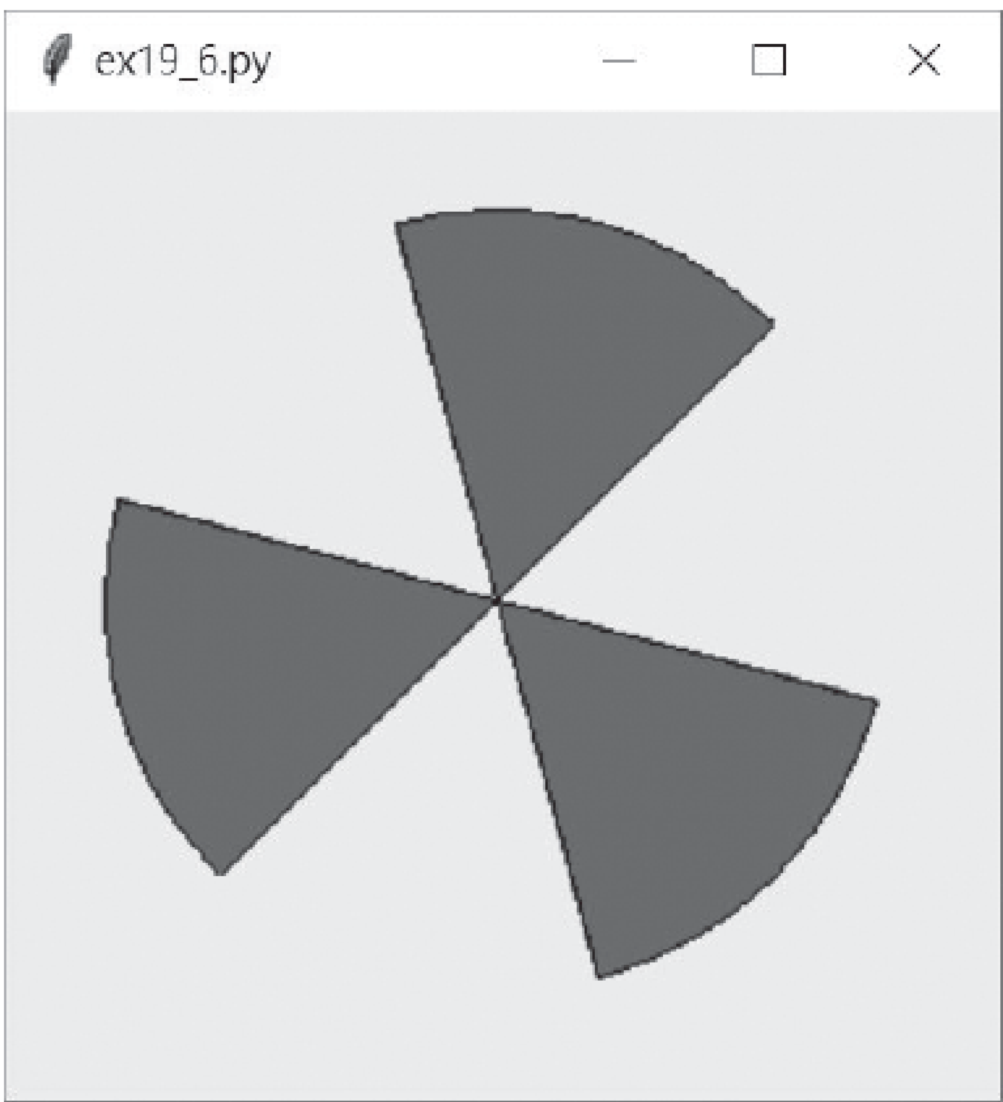
4. 写一个程序，可以显示走马灯信息。(19-3 节)



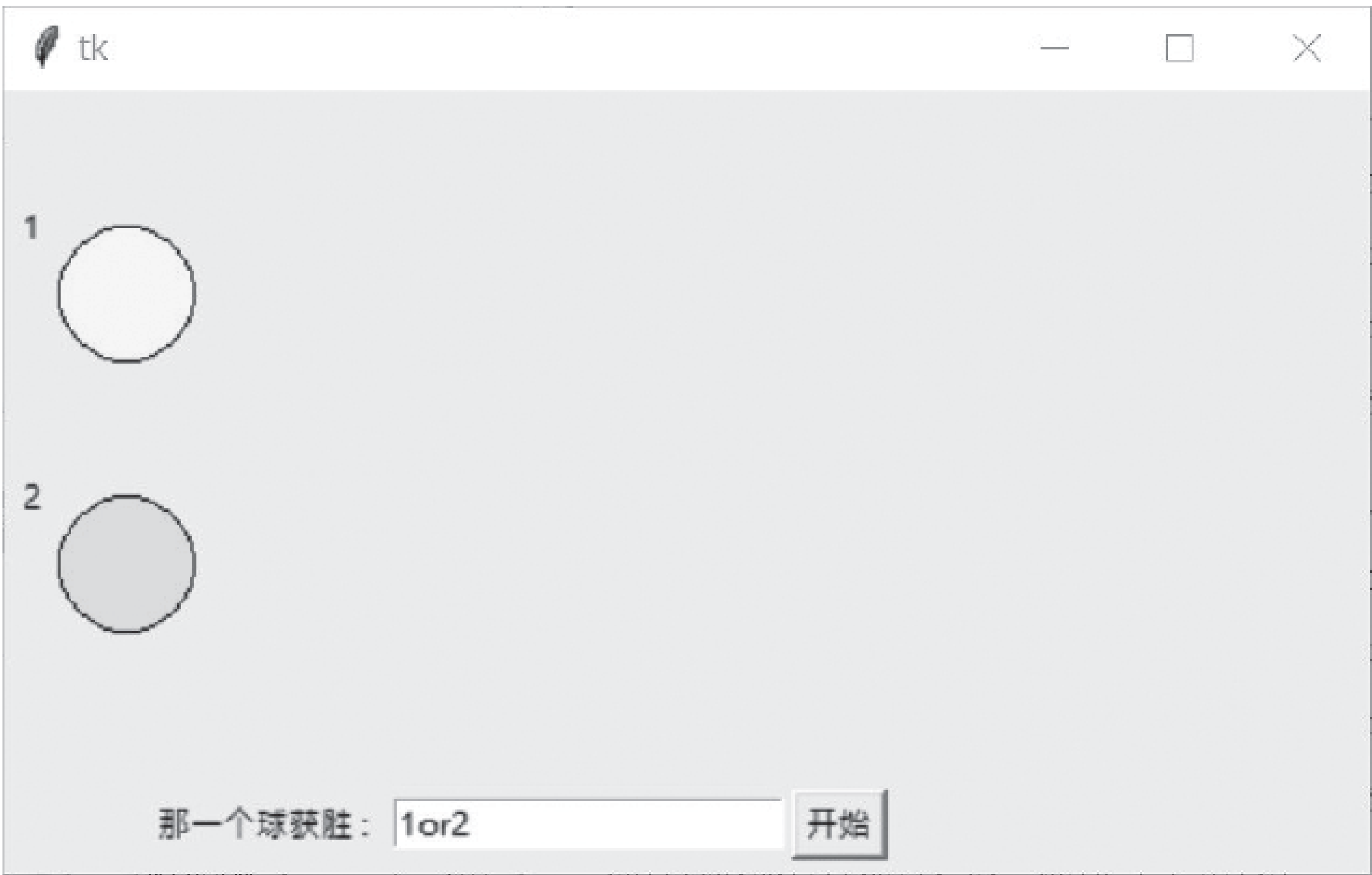
5. 写一个程序，当按键盘的上、下、左、右箭头键时，可以绘制线条。(19-3 节)



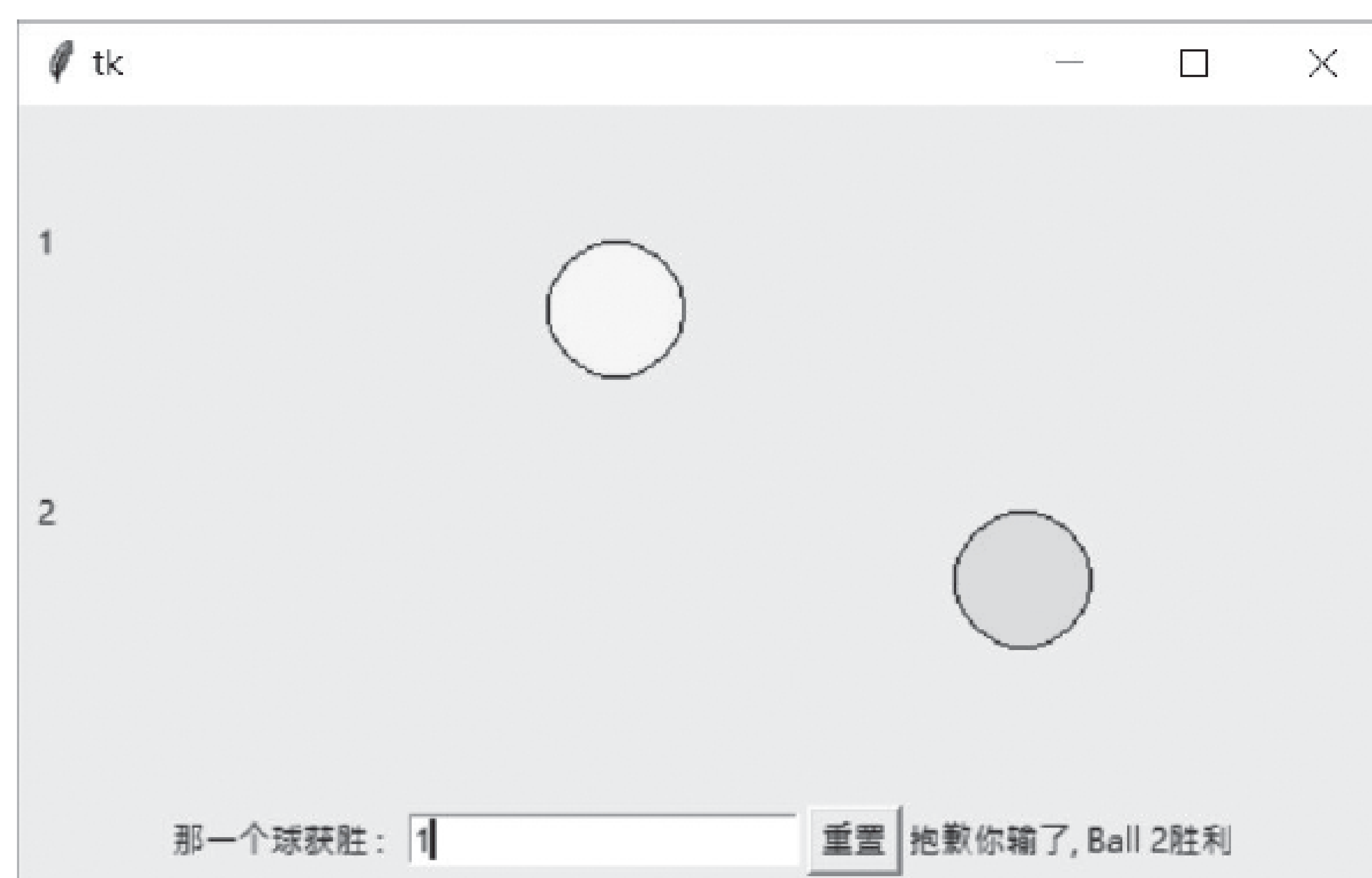
6. 绘制含有 3 片叶子的风扇，窗口的宽度与高度都是 300，风扇的半径是 120，其他如风扇颜色与转动细节则可以自行发挥。(19-3 节)



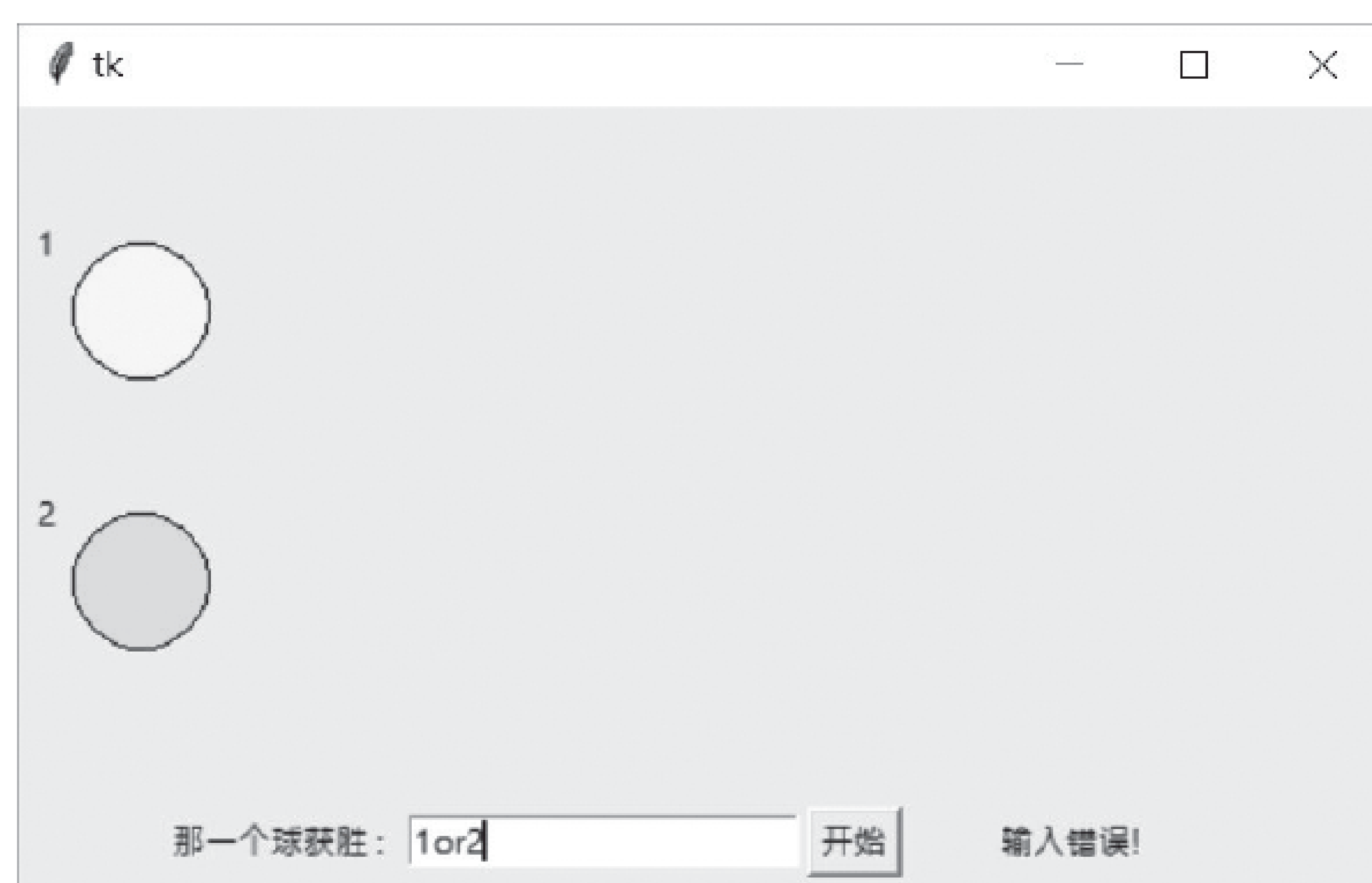
7. 重新设计程序实例 ch19_19.py，输出字符串让玩家由屏幕输入猜哪一个球跑得快，每次移动时都让计算机有 60% 移动的概率。下列是开始画面。(19-3 节)



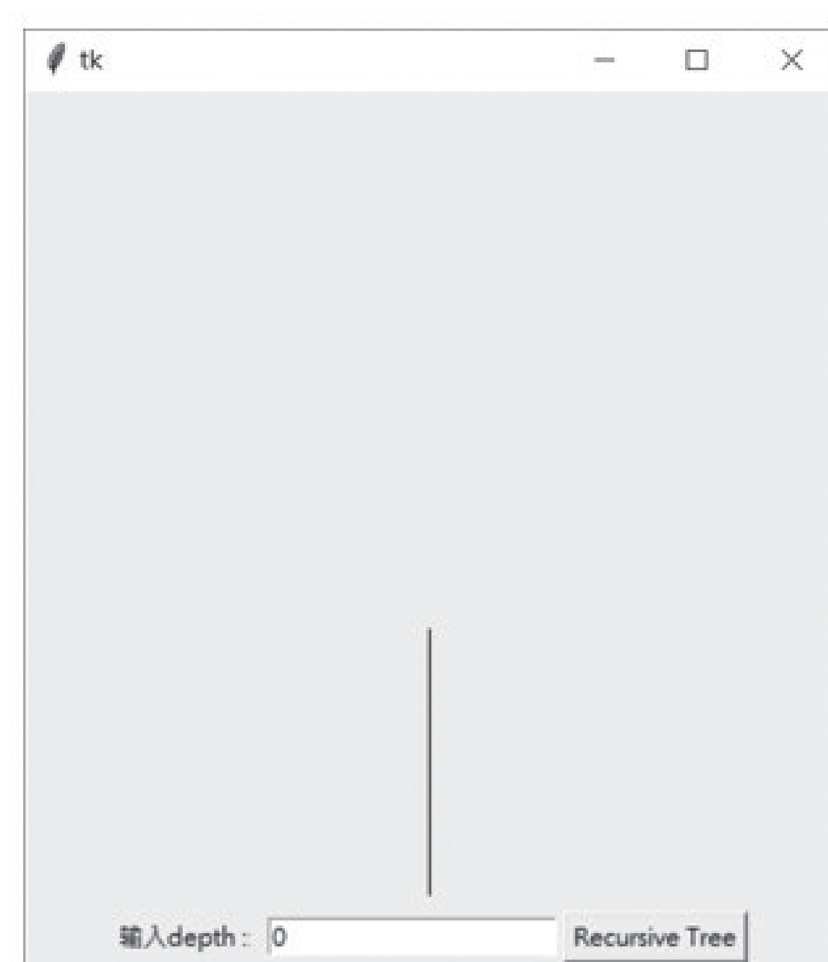
下列是选择 1 号球胜利，结果是 2 号球胜利的画面。

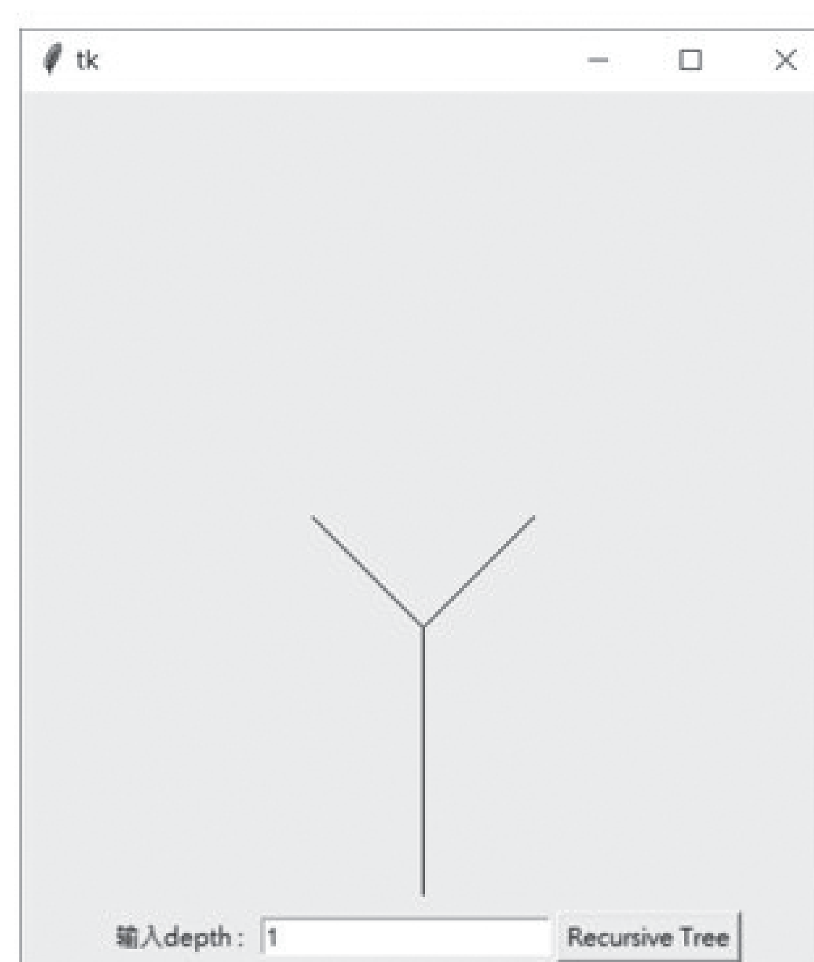


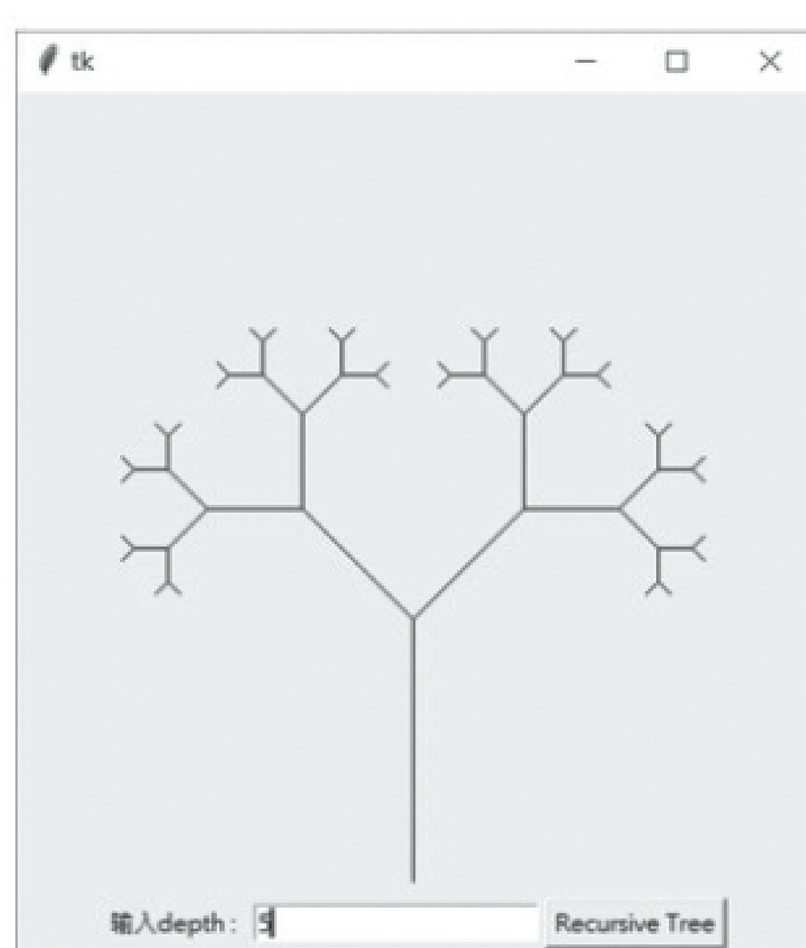
下列是输入错误的画面。

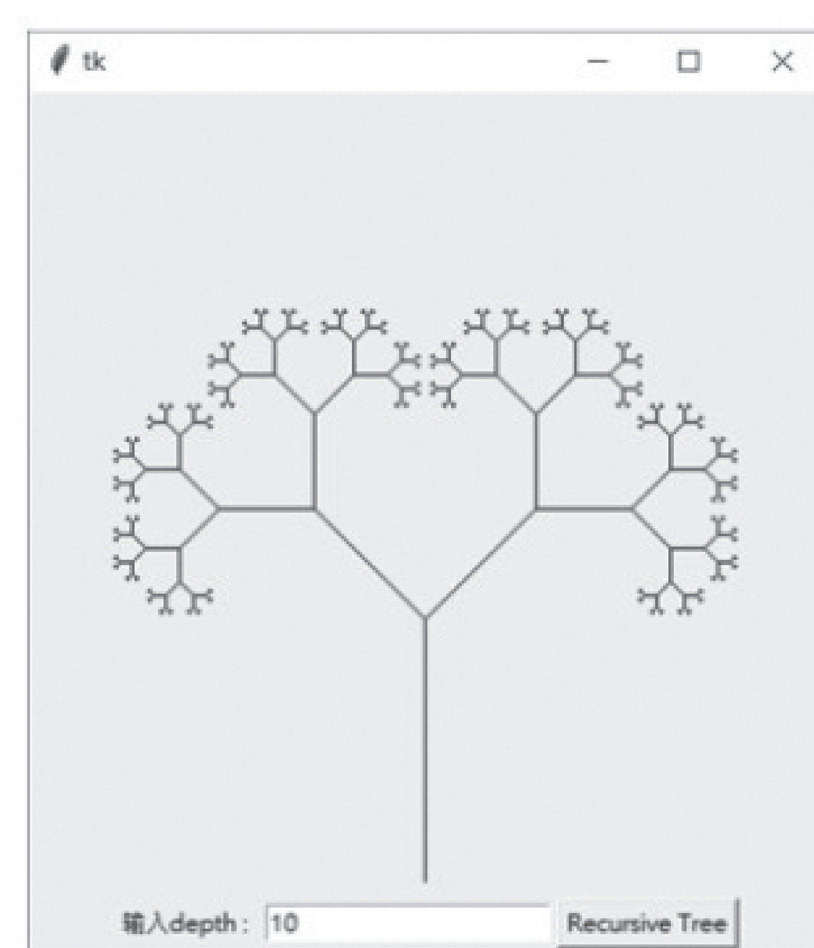


8. 参考 ch19_29.py，绘制一个递归树 Recursive Tree，假设树的分支是直角，下一层的树枝长度是前一层的 0.6 倍，下列是不同深度（depth）的递归树。(19-5 节)









20

第 2 0 章

数据图表的设计

本章摘要

20-1 绘制简单的折线图
20-2 绘制散点图 scatter( )
20-3 Numpy 模块
20-4 随机数的应用
20-5 绘制多个图表
20-6 直方图的制作
20-7 圆饼图的制作 pie( )
20-8 图表显示中文
20-9 专题——股市数据读取与图表制作

本章所讲述的重点是数据图形的绘制，所使用的工具是 matplotlib 绘图库模块，使用前需先安装：

```
pip install matplotlib
```

matplotlib 是一个庞大的绘图库模块，本章只导入其中的 pyplot 子模块就可以完成许多图表绘制，如下所示，未来就可以使用 plt 调用相关的方法了。

```
import matplotlib.pyplot as plt
```

本章将讲述 matplotlib 的重点，更完整的使用说明可以参考下列网站。

http://matplotlib.org

20-1　绘制简单的折线图

本节将从最简单的折线图开始介绍。

20-1-1　显示绘制的图形 show( )

这个 show( ) 方法主要是显示所绘制的图形，当绘制图形完成后，可以调用此方法。

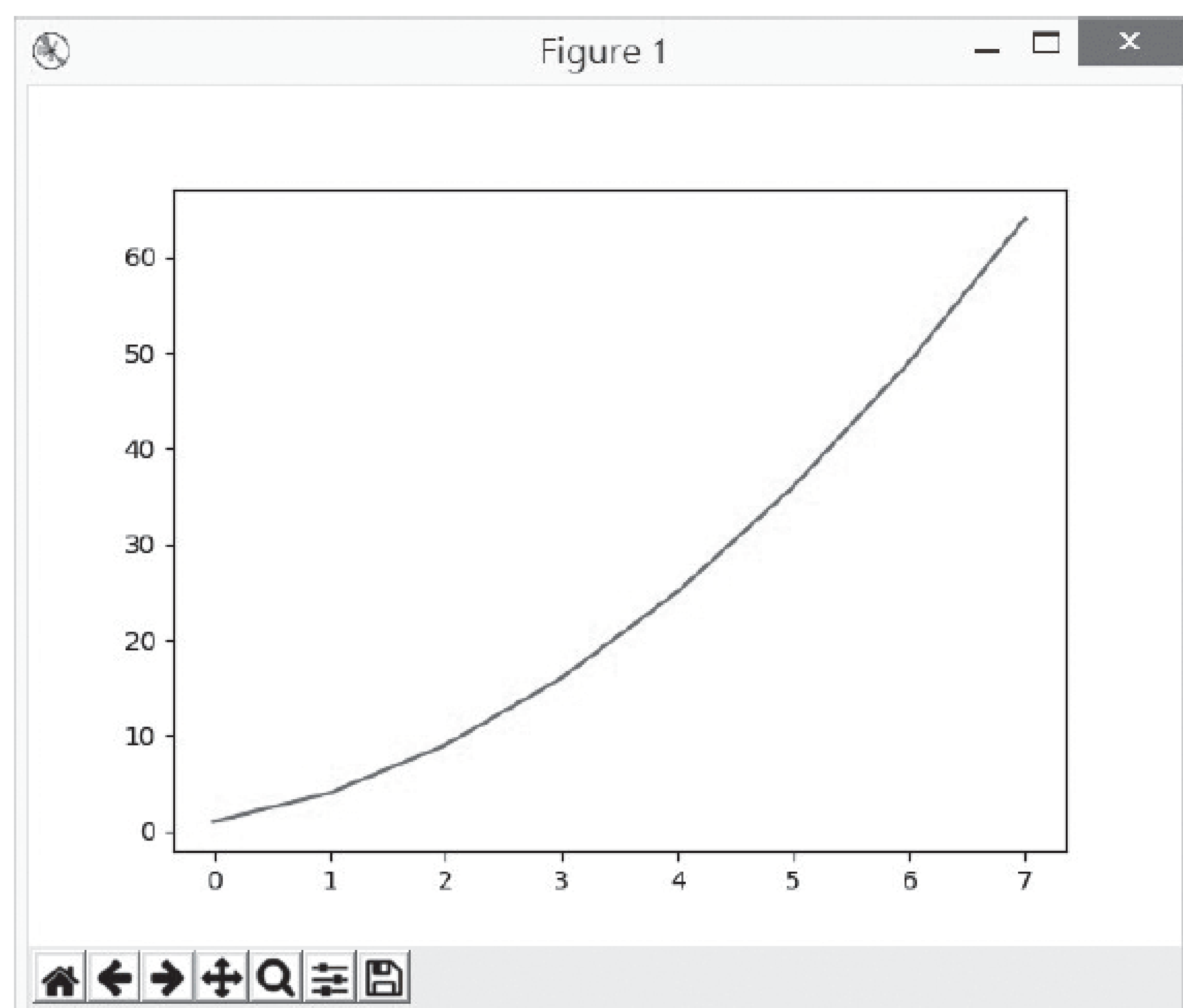
20-1-2　画线 plot( )

应用方式是将含数据的列表当作参数传给 plot( )，列表内的数据会被视为 y 轴的值，x 轴的值会依列表值的索引位置自动产生。

程序实例 ch20_1.py：绘制折线的应用。square[ ] 列表中有 8 个数据代表 y 轴值，这些数据基本上是 x 轴索引 0 ～ 7 的平方值序列。

```
1  # ch20_1.py
2  import matplotlib.pyplot as plt
3
4  squares = [1, 4, 9, 16, 25, 36, 49, 64]
5  plt.plot(squares)        # 列表 squares数据是y轴的值
6  plt.show()
```

执行结果

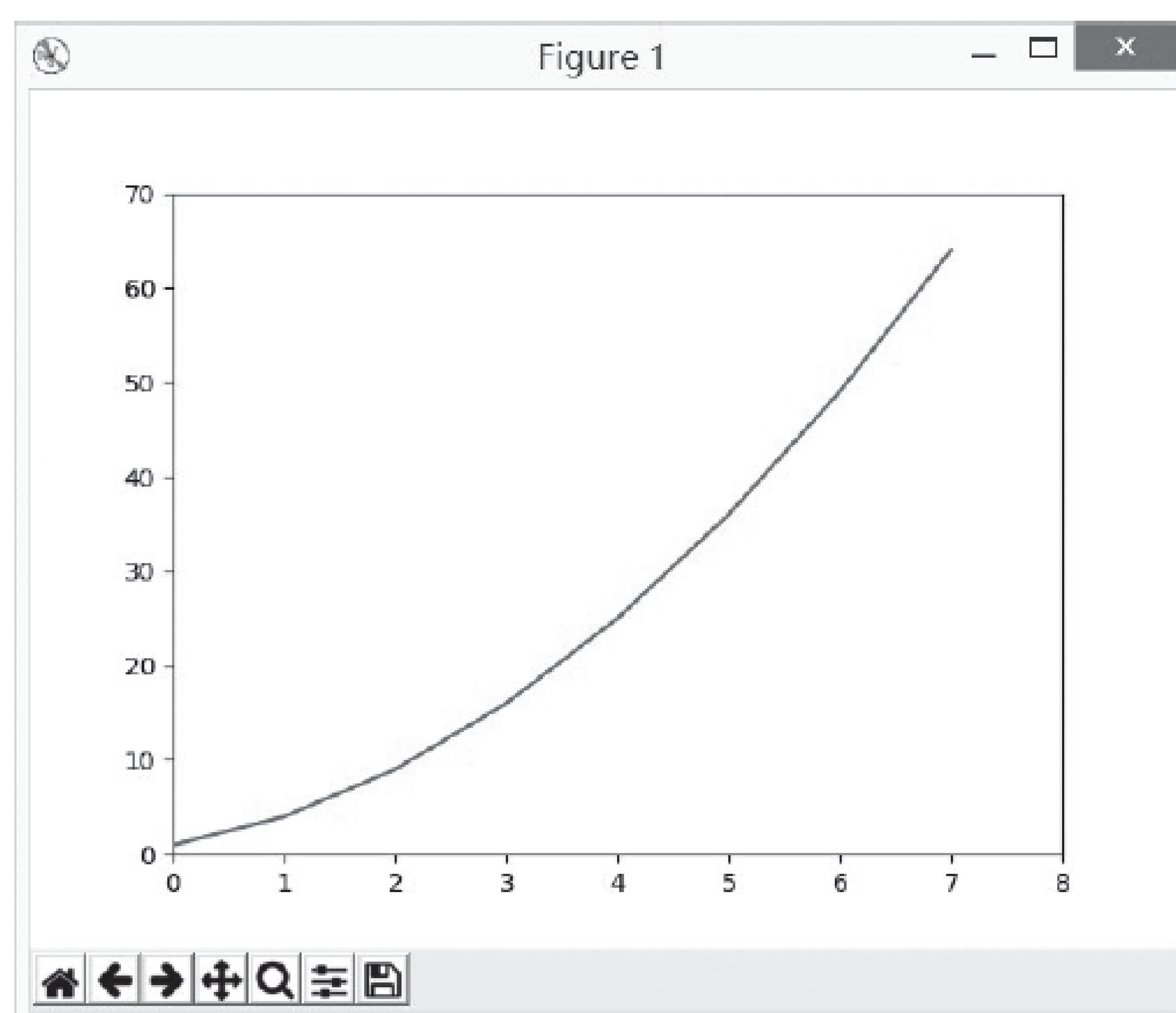


从上述执行结果可以看到，左下角的轴刻度不是（0,0），可以使用 axis( ) 设置 x 轴与 y 轴的最小和最大刻度。

程序实例 ch20_1_1.py：重新设计 ch20_1.py，将 x 轴刻度设为 0 ～ 8，y 轴刻度设为 0 ～ 70。

```
1  # ch20_1_1.py
2  import matplotlib.pyplot as plt
3
4  squares = [1, 4, 9, 16, 25, 36, 49, 64]
5  plt.plot(squares)          # 列表 squares数据是y轴的值
6  plt.axis([0, 8, 0, 70]) # x轴刻度0~8, y轴刻度0~70
7  plt.show()
```

执行结果



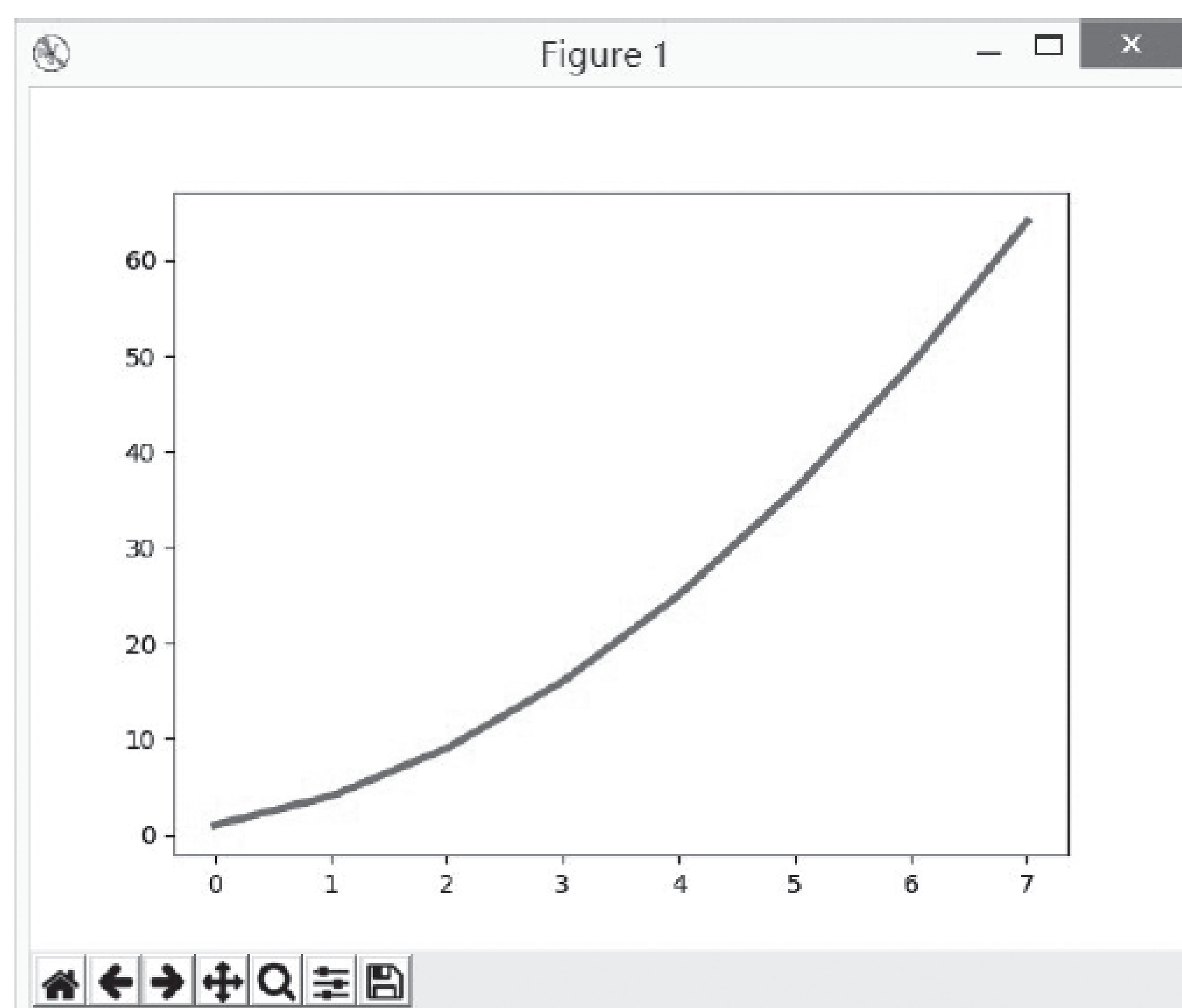
20-1-3 线条宽度 linewidth

使用 plot( ) 时默认线条宽度是 1，可以多加一个 linewidth（缩写是 lw）参数设置线条的粗细。

程序实例 ch20_2.py：设置线条宽度是 3。

```
1  # ch20_2.py
2  import matplotlib.pyplot as plt
3
4  squares = [1, 4, 9, 16, 25, 36, 49, 64]
5  plt.plot(squares, linewidth=3)
6  plt.show()
```

执行结果



20-1-4　标题的显示

目前 matplotlib 模块默认不支持中文显示，笔者将在 20-8 节讲解如何更改字体，让图表可以显示中文。下面是图表几个重要的方法。

title( )：图表标题。

xlabel( )：x 轴标题。

ylabel( )：y 轴标题。

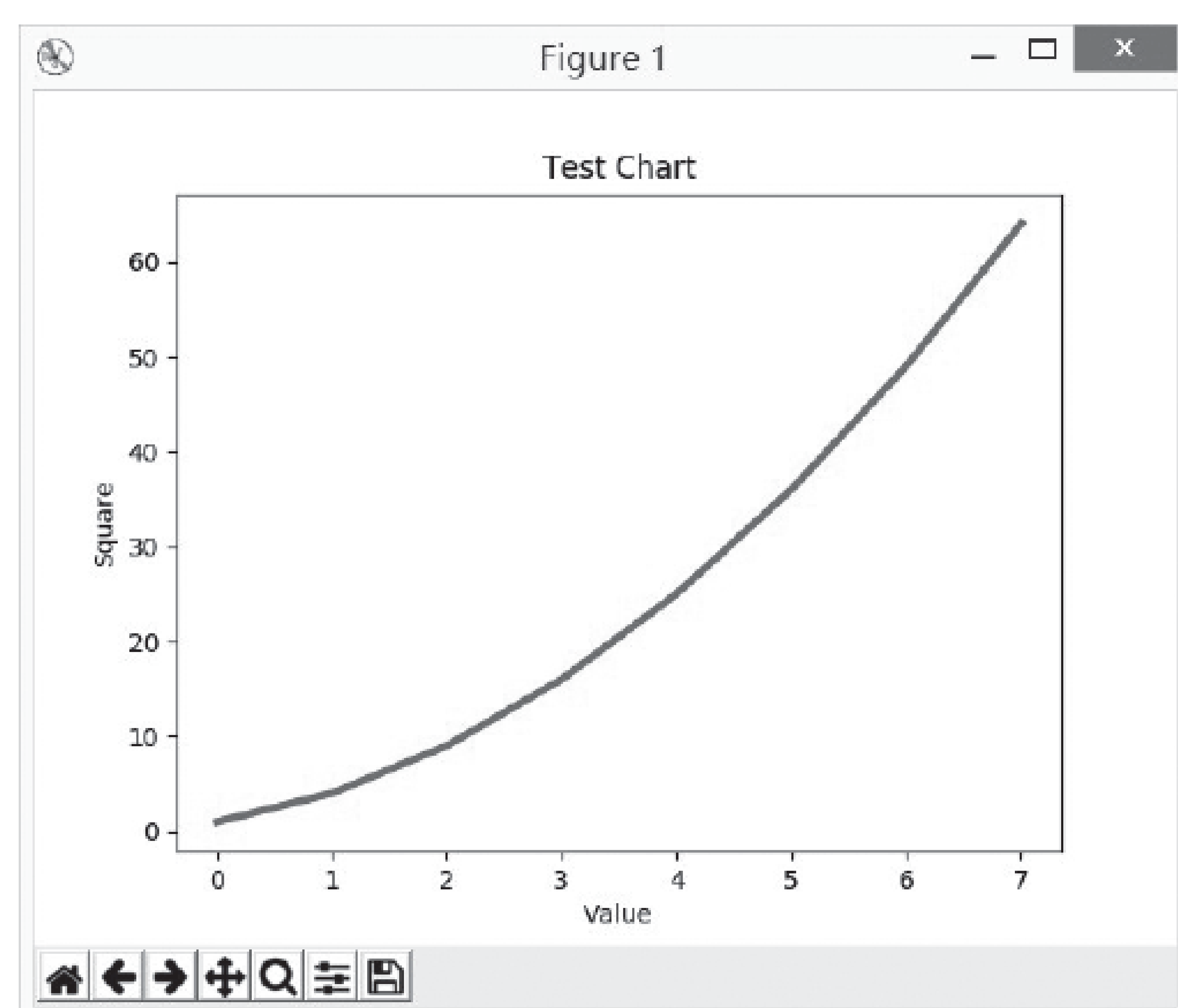
上述方法可以显示默认大小是 12 的字体，语法如下。

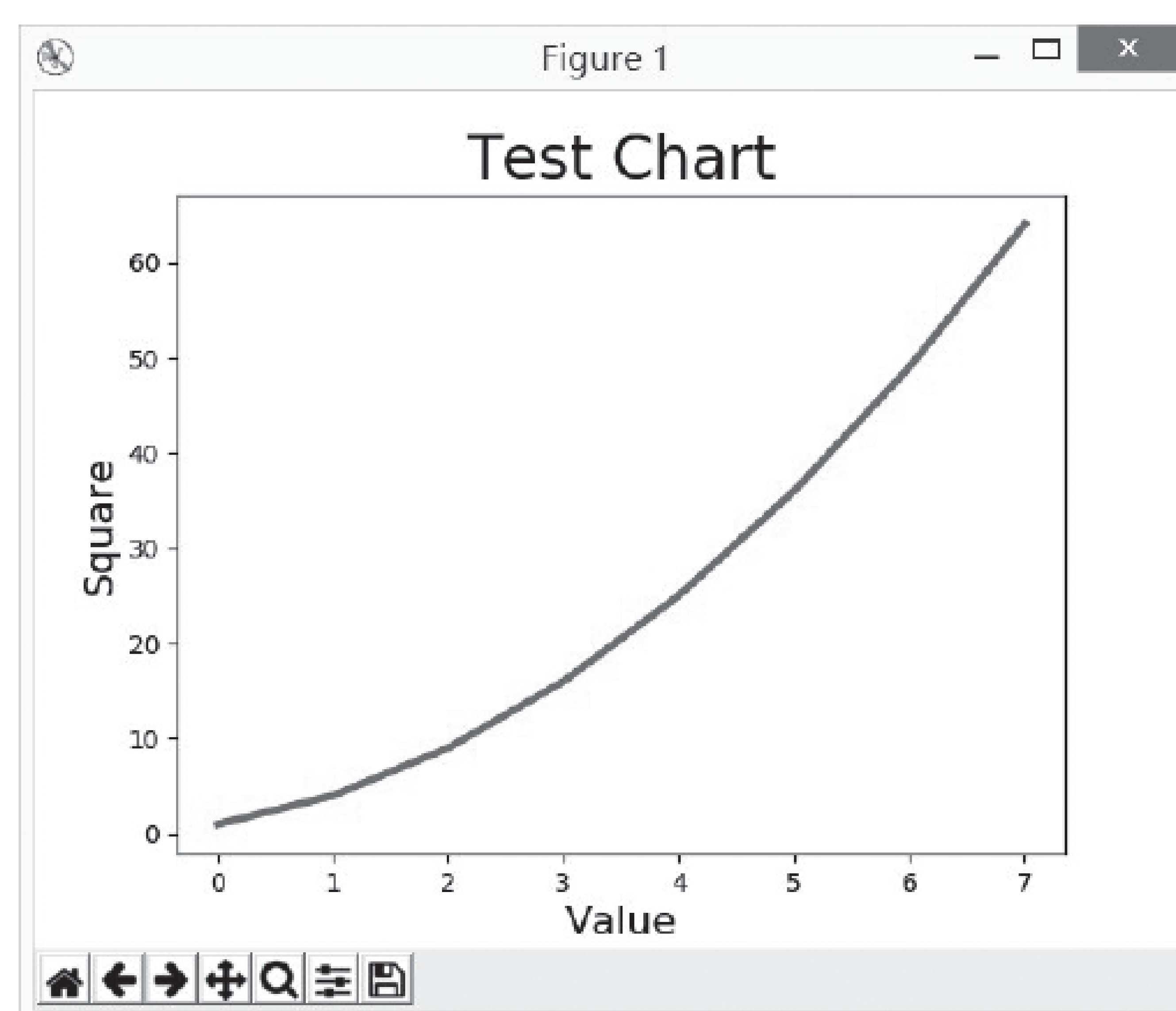
title（标题名称，fontsize= 字号）　　# 同时可用于 xlabel（ ）和 ylabel（ ）

程序实例 ch20_3.py：使用默认字号为图表与 x 轴及 y 轴建立标题。

```
1  # ch20_3.py
2  import matplotlib.pyplot as plt
3
4  squares = [1, 4, 9, 16, 25, 36, 49, 64]
5  plt.plot(squares, linewidth=3)
6  plt.title("Test Chart")
7  plt.xlabel("Value")
8  plt.ylabel("Square")
9  plt.show()
```

执行结果　可参考下方左图。





程序实例 ch20_4.py：设置图表标题字号为 24，x 轴与 y 轴标题字号为 16。

```
1  # ch20_4.py
2  import matplotlib.pyplot as plt
3
4  squares = [1, 4, 9, 16, 25, 36, 49, 64]
5  plt.plot(squares, linewidth=3)
6  plt.title("Test Chart", fontsize=24)
7  plt.xlabel("Value", fontsize=16)
8  plt.ylabel("Square", fontsize=16)
9  plt.show()
```

执行结果　可参考上方右图。

20-1-5　坐标轴刻度的设置

在设计图表时可以使用 tick_params( ) 设置坐标轴的**刻度大小**、**颜色**以及应用范围。

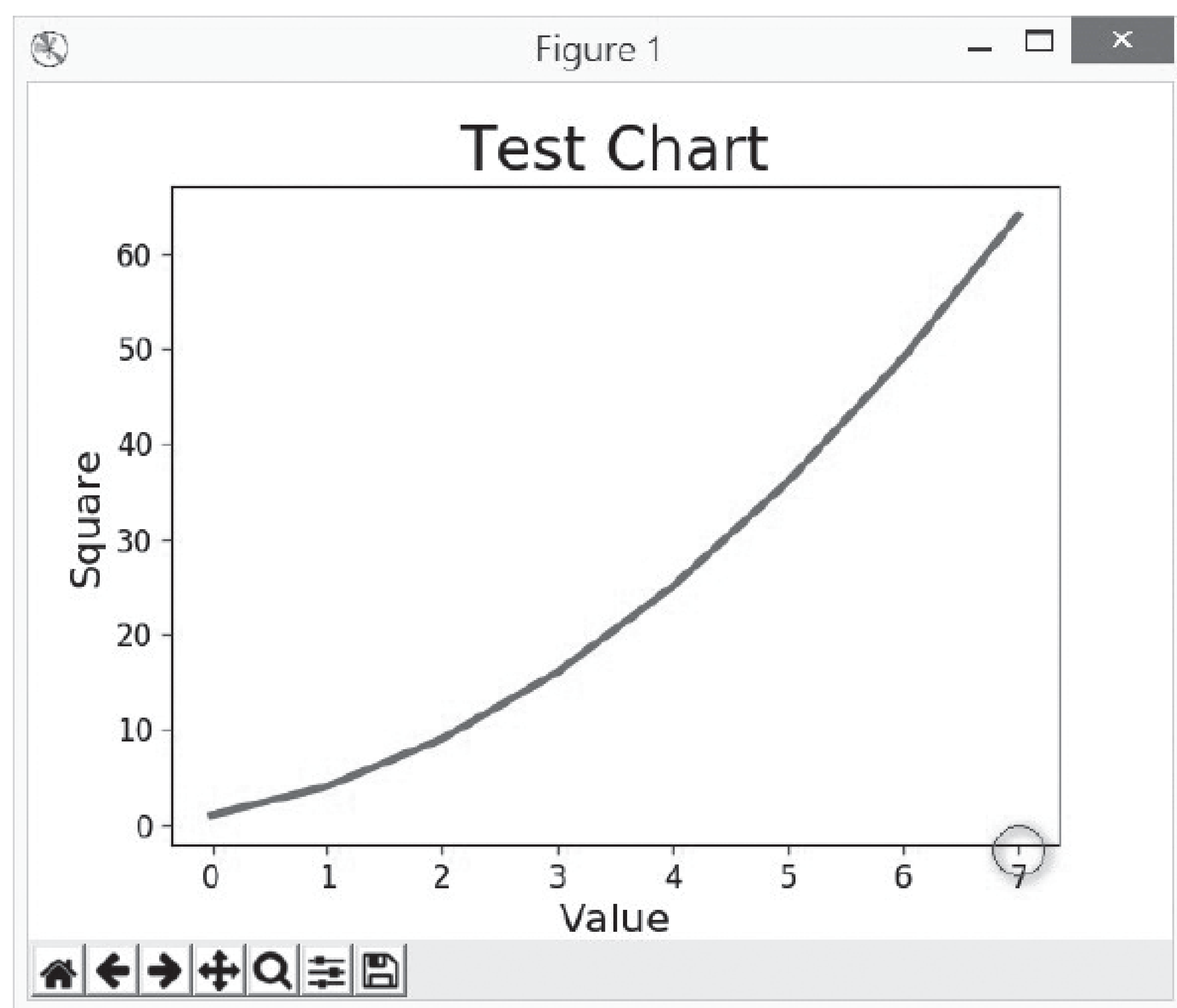
tick_params(axis='**xx**', labelsize=**xx**, color='xx')　　# labelsize 的 **xx** 代表刻度大小

如果 axis 的 xx 是 both，代表应用到 x 轴和 y 轴；如果 xx 是 x，代表应用到 x 轴；如果 xx 是 y，代表应用到 y 轴。color 则是设置刻度的线条颜色，例如，red 代表红色。20-1-8 节会有颜色表。

程序实例 ch20_5.py：使用不同刻度与颜色的应用。

```
1   # ch20_5.py
2   import matplotlib.pyplot as plt
3
4   squares = [1, 4, 9, 16, 25, 36, 49, 64]
5   plt.plot(squares, linewidth=3)
6   plt.title("Test Chart", fontsize=24)
7   plt.xlabel("Value", fontsize=16)
8   plt.ylabel("Square", fontsize=16)
9   plt.tick_params(axis='both', labelsize=12, color='red')
10  plt.show()
```

执行结果



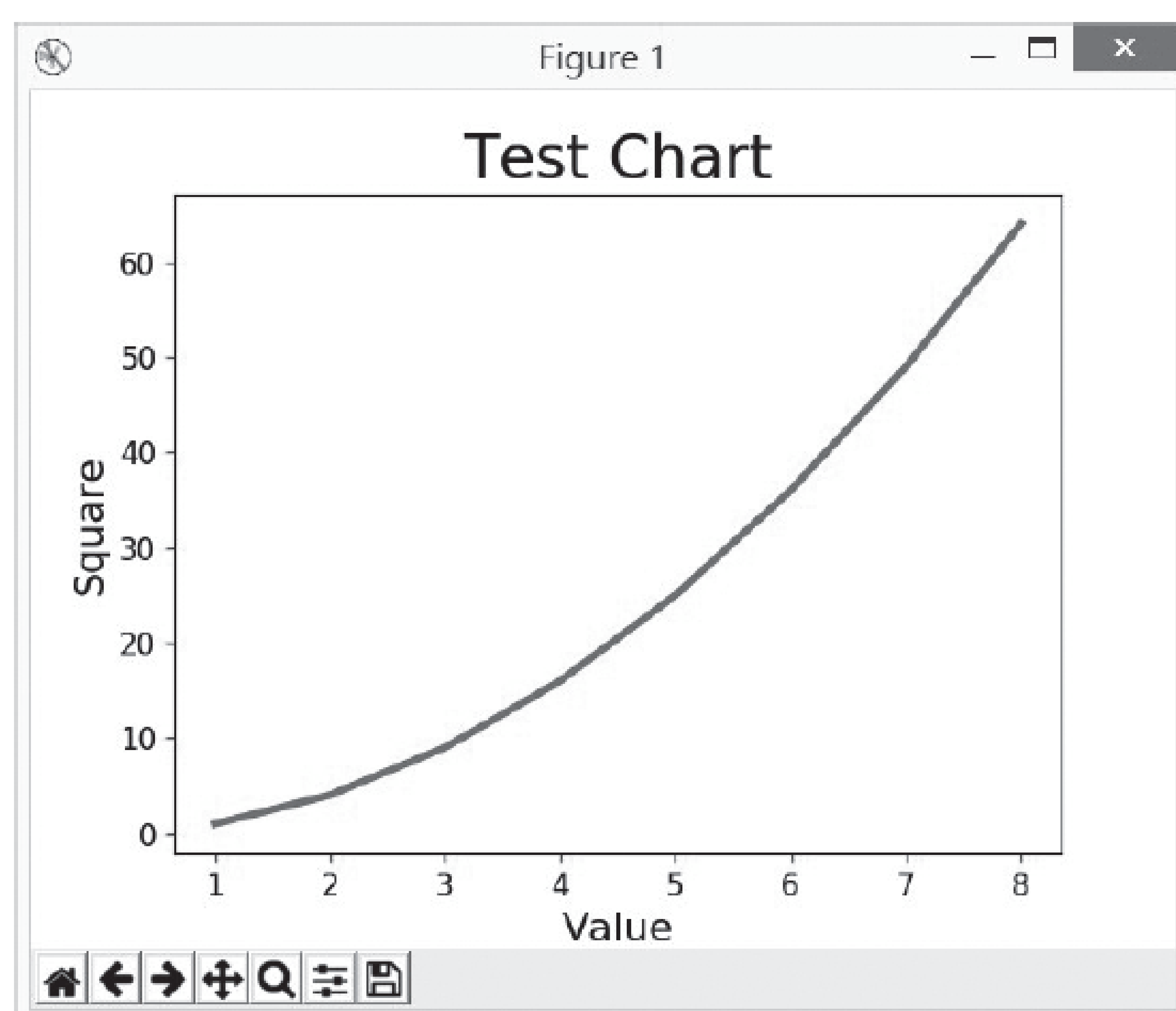
20-1-6　修订图表的起始值

从上图可以看到平方列表的值是有 8 个数据，依照 Python 语法起始数字是 0，所以到 7 结束。但是日常生活中，报表数字通常是从 1 开始的，为了做这个修订，可以再增加一个列表，这个列表主要是设置数值索引，细节可参考下列实例的第 5 行 seq。

程序实例 ch20_6.py：修订图表的起始值，使 x 轴的刻度从 1 开始。

```
1   # ch20_6.py
2   import matplotlib.pyplot as plt
3
4   squares = [1, 4, 9, 16, 25, 36, 49, 64]
5   seq = [1,2,3,4,5,6,7,8]
6   plt.plot(seq, squares, linewidth=3)
7   plt.title("Test Chart", fontsize=24)
8   plt.xlabel("Value", fontsize=16)
9   plt.ylabel("Square", fontsize=16)
10  plt.tick_params(axis='both', labelsize=12, color='red')
11  plt.show()
```

执行结果



20-1-7　多组数据的应用

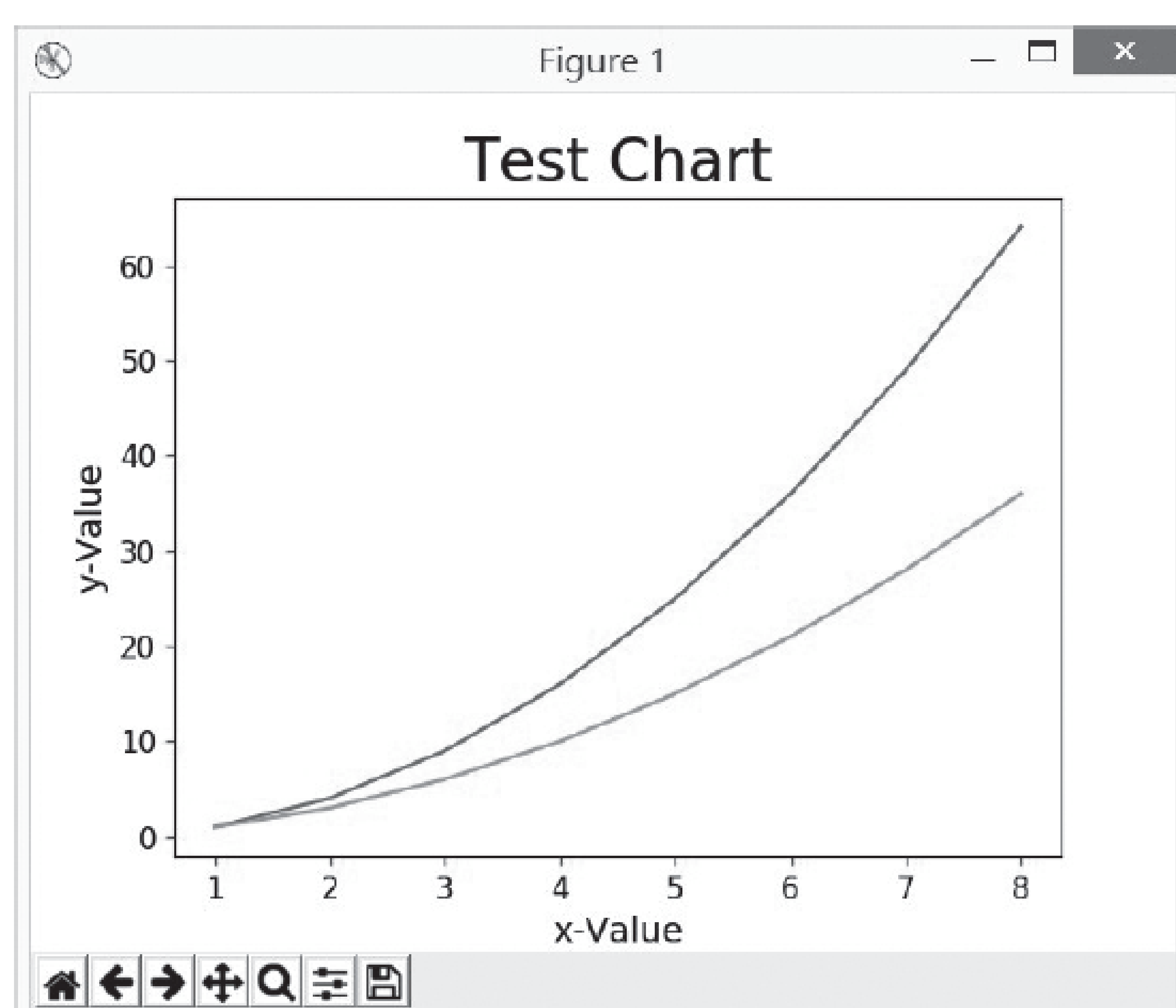
目前所有的图表都是只有一组数据，其实可以扩充多组数据，只要在 plot( ) 内增加数据列表参数即可。此时 plot( ) 的参数如下。

plot(seq, 第一组数据 , seq, 第二组数据 , …)　# seq 的概念可以参考 20-1-6 节

程序实例 ch20_7：设计多组数据图的应用。

```
1  # ch20_7.py
2  import matplotlib.pyplot as plt
3
4  data1 = [1, 4, 9, 16, 25, 36, 49, 64]         # data1线条
5  data2 = [1, 3, 6, 10, 15, 21, 28, 36]         # data2线条
6  seq = [1,2,3,4,5,6,7,8]
7  plt.plot(seq, data1, seq, data2)              # data1&data2线条
8  plt.title("Test Chart", fontsize=24)
9  plt.xlabel("x-Value", fontsize=14)
10 plt.ylabel("y-Value", fontsize=14)
11 plt.tick_params(axis='both', labelsize=12, color='red')
12 plt.show()
```

执行结果



上述以不同颜色显示线条是系统默认，也可以自定义线条色彩。

20-1-8 线条色彩与样式

如果想设置线条色彩，可以在 plot( ) 内增加下列参数设置，下列是常见的色彩表。

色彩字符	色彩说明
'b'	blue（蓝色）
'c'	cyan（青色）
'g'	green（绿色）
'k'	black（黑色）
'm'	magenta（品红）
'r'	red（红色）
'w'	white（白色）
'y'	yellow（黄色）

下列是常见的样式表单。

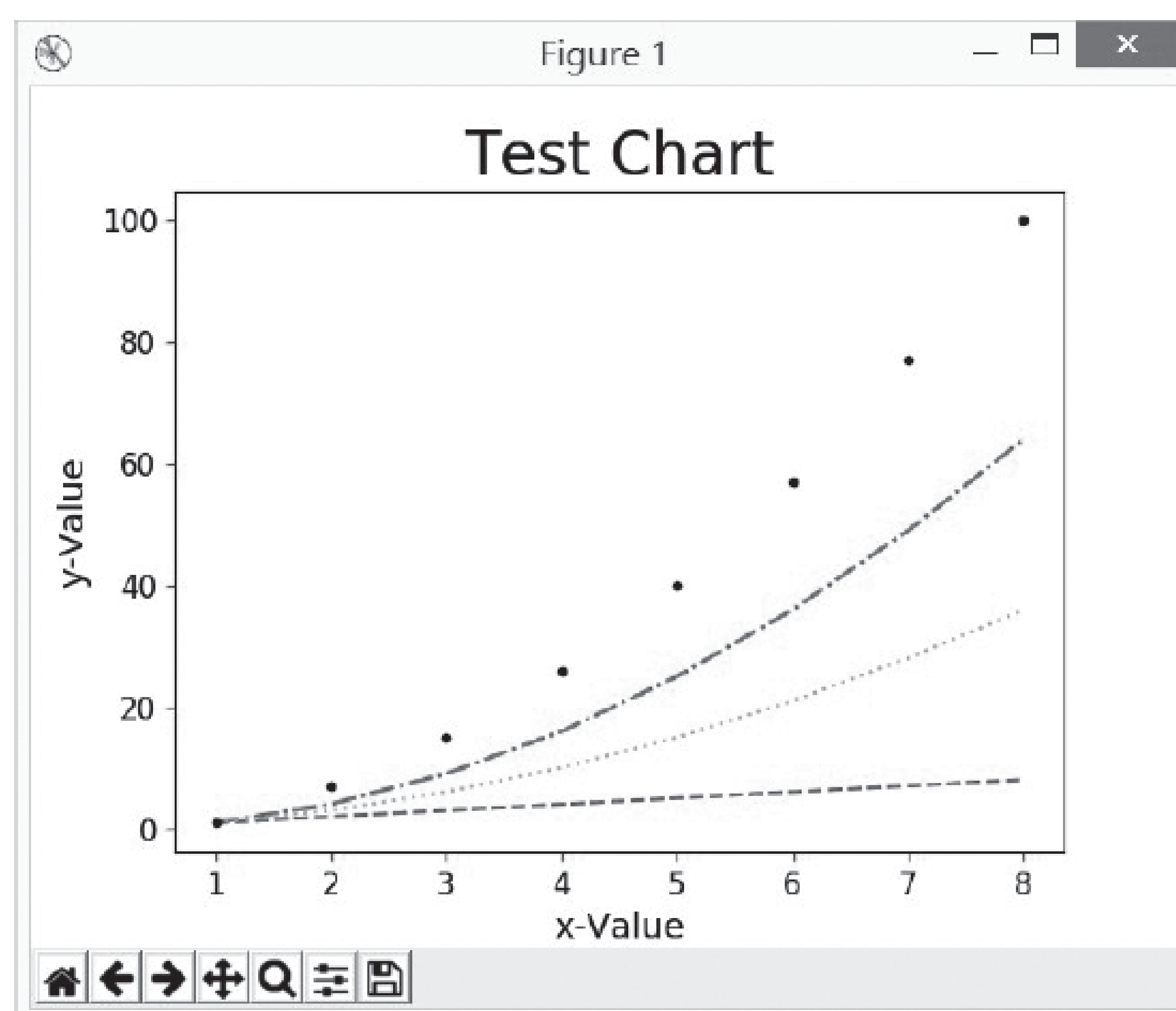
字符	说明
'-' 或 "solid'	默认实线
'- -' 或 'dashed'	虚线
'-.' 或 'dashdot'	虚点线
':' 或 'dotted'	点线
'.'	点标记
','	像素标记
'o'	圆标记
'v'	反三角标记
'^'	三角标记
'<'	左三角形
'>'	右三角形
's'	方形标记
'p'	五角标记
'*'	星星标记
'+'	加号标记
'-'	减号标记
'x'	X 标记
'H'	六边形 1 标记
'h'	六边形 2 标记

上述样式可以混合使用，例如，'r-.' 代表红色虚点线。

程序实例 ch20_8.py：采用不同色彩与线条样式绘制图表。

```
1   # ch20_8.py
2   import matplotlib.pyplot as plt
3
4   data1 = [1, 2, 3, 4, 5, 6, 7, 8]                # data1线条
5   data2 = [1, 4, 9, 16, 25, 36, 49, 64]           # data2线条
6   data3 = [1, 3, 6, 10, 15, 21, 28, 36]           # data3线条
7   data4 = [1, 7, 15, 26, 40, 57, 77, 100]         # data4线条
8
9   seq = [1, 2, 3, 4, 5, 6, 7, 8]
10  plt.plot(seq, data1, 'g--', seq, data2, 'r-.', seq, data3, 'y:', seq, data4, 'k.')
11  plt.title("Test Chart", fontsize=24)
12  plt.xlabel("x-Value", fontsize=14)
13  plt.ylabel("y-Value", fontsize=14)
14  plt.tick_params(axis='both', labelsize=12, color='red')
15  plt.show()
```

执行结果

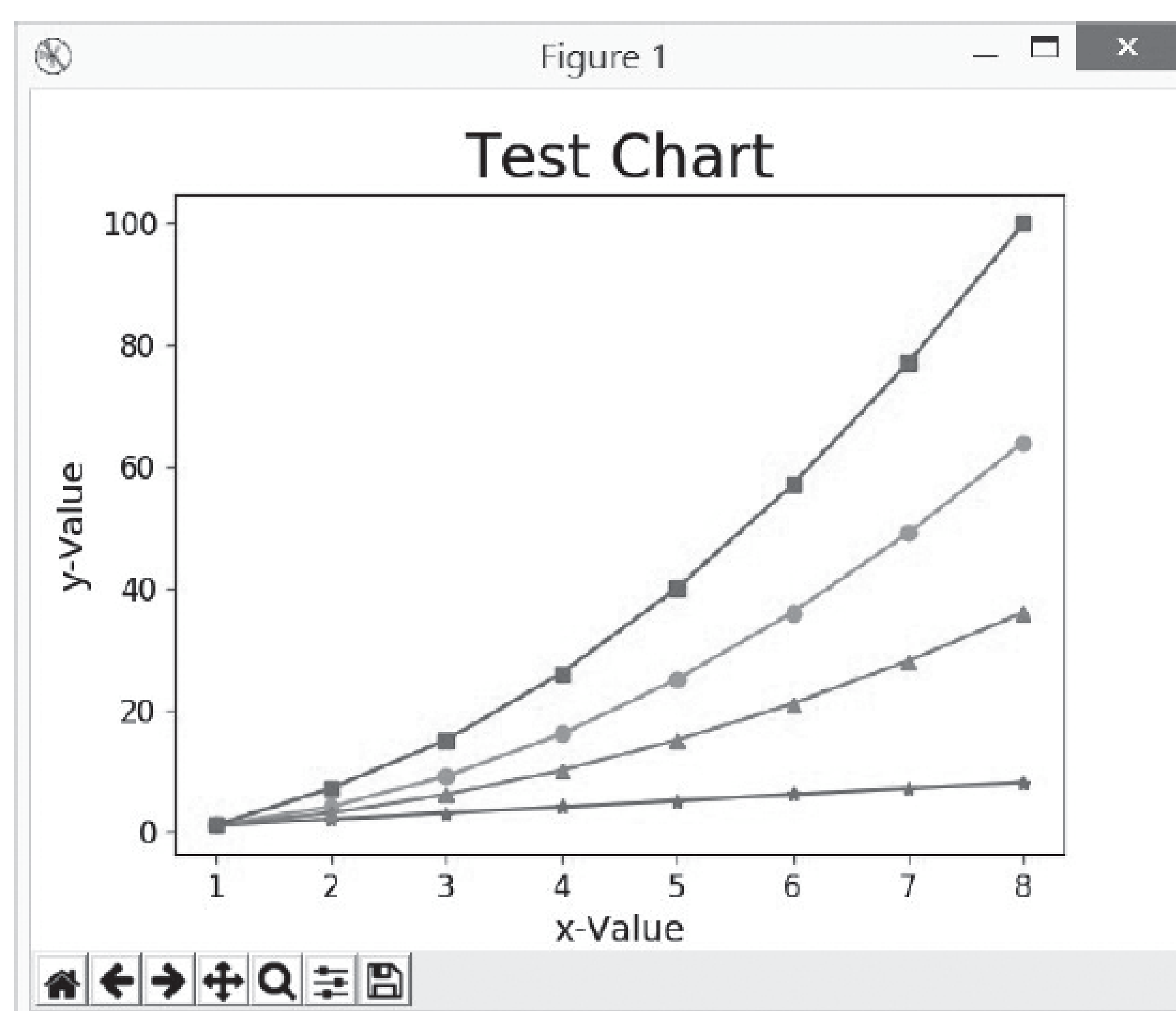


在上述第 10 行最右边 'k.' 代表绘制黑点而不是绘制线条，读者可以使用不同颜色绘制散点图，20-2 节还会介绍另一个方法 scatter( ) 绘制散点图。上述格式的应用是很灵活的，如果我们使用 '-*' 可以绘制线条，同时在指定点加上星星标记。**注**：如果没有设置颜色，系统会自行配置颜色。

程序实例 ch20_9.py：重新设计 ch20_8.py 绘制线条，同时为各个点加上标记。

```
10  plt.plot(seq, data1, '-*', seq, data2, '-o', seq, data3, '-^', seq, data4, '-s')
```

执行结果



20-1-9 刻度设计

目前所有图表中的 x 轴和 y 轴的刻度都是用 plot( ) 方法针对所输入的参数采用默认值设置，请先参考下列实例。

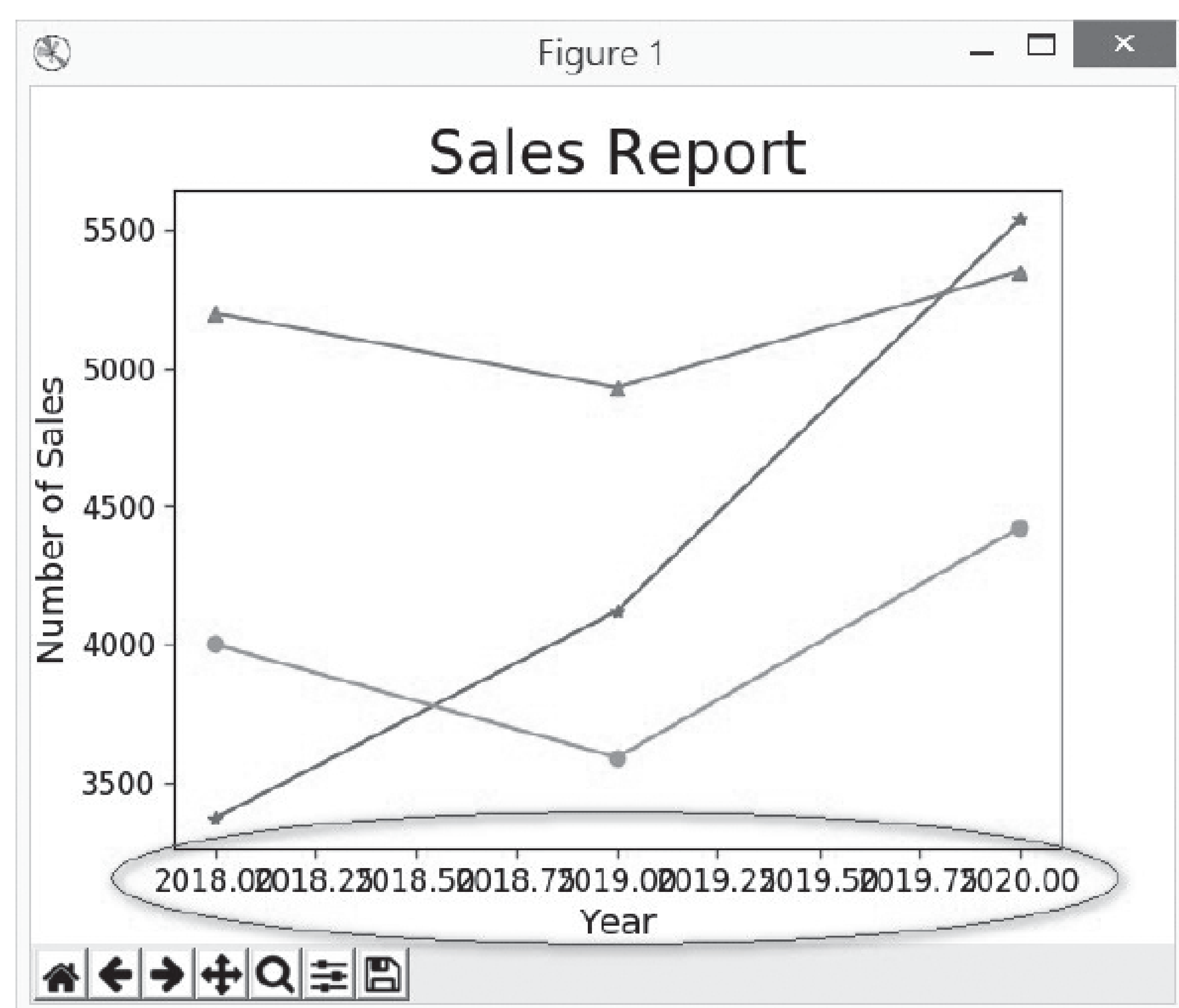
程序实例 ch20_10.py：假设 3 大品牌车辆 2018—2020 年的销售数据如下。

```
Benz      3367   4120   5539
BMW 4000  3590   4423
Lexus     5200   4930   5350
```

请将上述数据绘制成图表。

```
1  # ch20_10.py
2  import matplotlib.pyplot as plt
3
4  Benz = [3367, 4120, 5539]                  # Benz线条
5  BMW = [4000, 3590, 4423]                   # BMW线条
6  Lexus = [5200, 4930, 5350]                 # Lexus线条
7
8  seq = [2018, 2019, 2020]                   # 年份
9  plt.plot(seq, Benz, '-*', seq, BMW, '-o', seq, Lexus, '-^')
10 plt.title("Sales Report", fontsize=24)
11 plt.xlabel("Year", fontsize=14)
12 plt.ylabel("Number of Sales", fontsize=14)
13 plt.tick_params(axis='both', labelsize=12, color='red')
14 plt.show()
```

执行结果

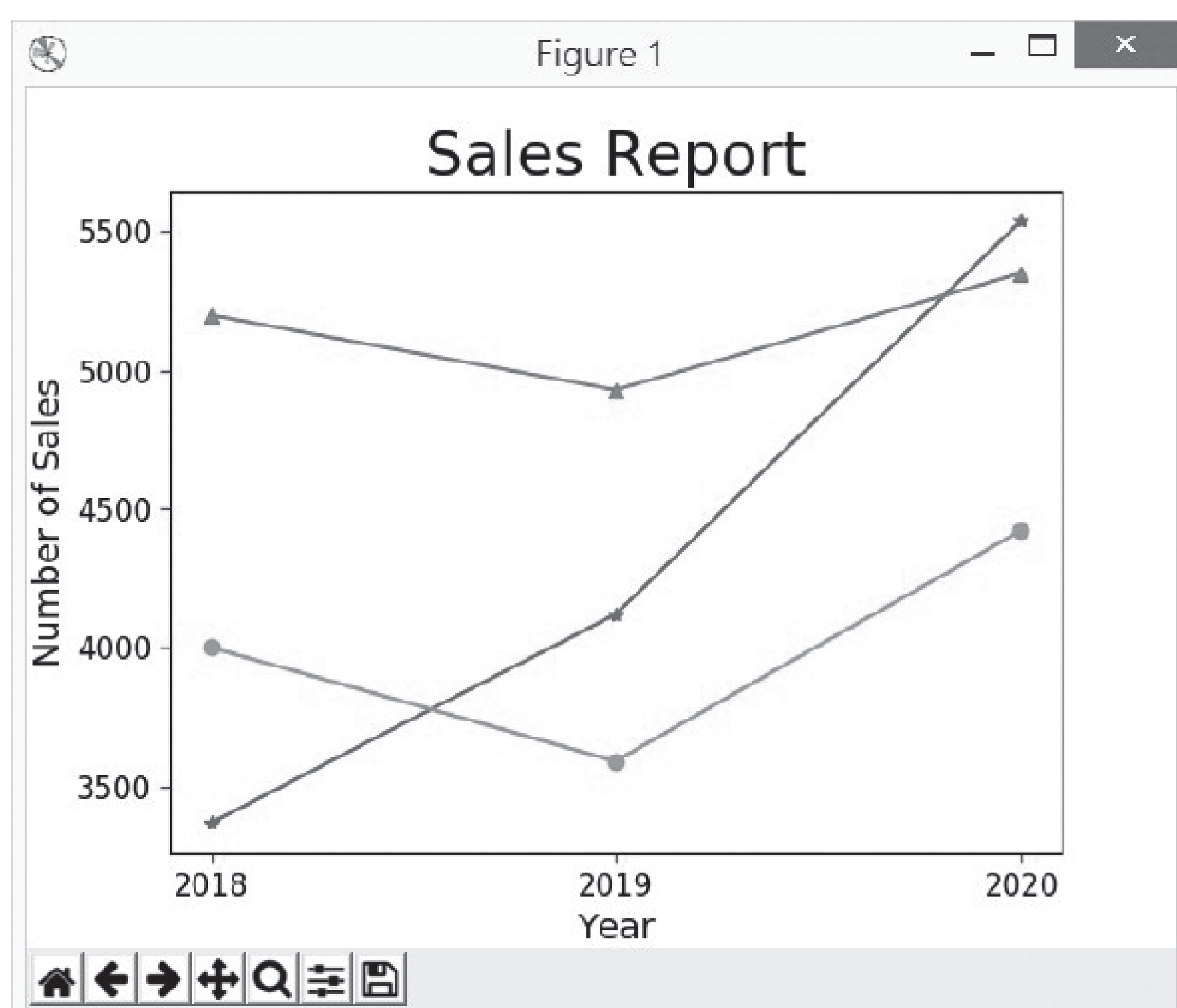


上述程序最大的遗憾是 x 轴的刻度，对我们而言，其实只要有 2018、2019、2020 这 3 个年份的刻度即可，还好可以使用 pyplot 模块的 xticks( ) 和 yticks( ) 分别设置 x 轴和 y 轴刻度，可参考下列实例。

程序实例 ch20_11.py：重新设计 ch20_10.py，自行设置刻度，这个程序的重点是第 9 行，将 seq 列表当作参数放在 plt.xticks( ) 内。

```
1  # ch20_11.py
2  import matplotlib.pyplot as plt
3
4  Benz = [3367, 4120, 5539]                # Benz线条
5  BMW = [4000, 3590, 4423]                 # BMW线条
6  Lexus = [5200, 4930, 5350]               # Lexus线条
7
8  seq = [2018, 2019, 2020]                 # 年份
9  plt.xticks(seq)                          # 设置x轴刻度
10 plt.plot(seq, Benz, '-*', seq, BMW, '-o', seq, Lexus, '-^')
11 plt.title("Sales Report", fontsize=24)
12 plt.xlabel("Year", fontsize=14)
13 plt.ylabel("Number of Sales", fontsize=14)
14 plt.tick_params(axis='both', labelsize=12, color='red')
15 plt.show()
```

执行结果



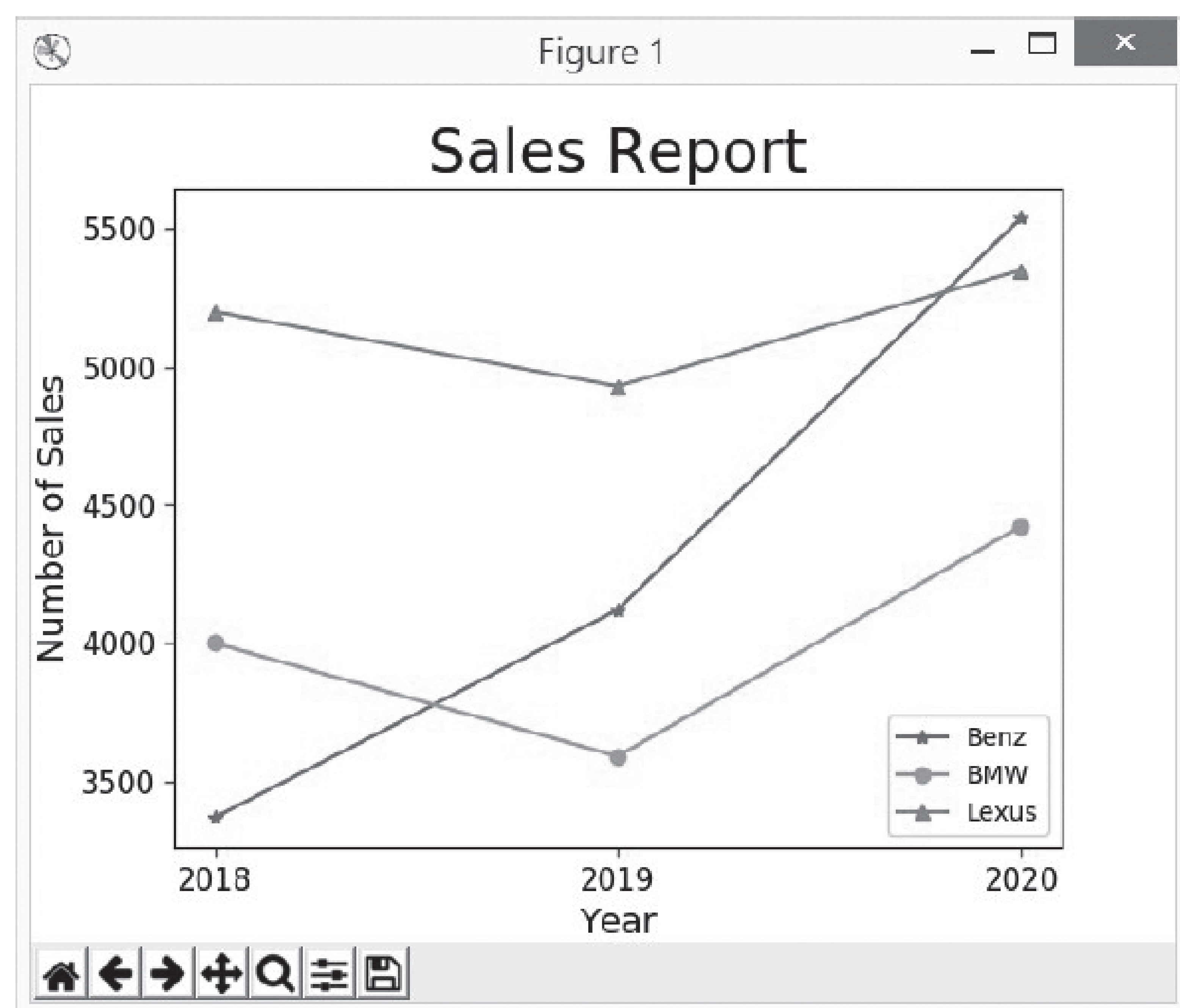
20-1-10　图例 legend( )

程序实例 ch20_1.py 所建立的图表已经很好了，缺点是缺乏各种线条代表的意义，也称图例（legend），下面将直接以实例说明。

程序实例 ch20_12.py：为 ch20_11.py 建立图例。

```
1  # ch20_12.py
2  import matplotlib.pyplot as plt
3
4  Benz = [3367, 4120, 5539]                # Benz线条
5  BMW = [4000, 3590, 4423]                 # BMW线条
6  Lexus = [5200, 4930, 5350]               # Lexus线条
7
8  seq = [2018, 2019, 2020]                 # 年份
9  plt.xticks(seq)                          # 设置x轴刻度
10 plt.plot(seq, Benz, '-*', label='Benz')
11 plt.plot(seq, BMW, '-o', label='BMW')
12 plt.plot(seq, Lexus, '-^', label='Lexus')
13 plt.legend(loc='best')
14 plt.title("Sales Report", fontsize=24)
15 plt.xlabel("Year", fontsize=14)
16 plt.ylabel("Number of Sales", fontsize=14)
17 plt.tick_params(axis='both', labelsize=12, color='red')
18 plt.show()
```

执行结果



这个程序最大的不同在第 10 ～ 12 行，以第 10 行说明。

```
plt.plot(seq, Benz, '-*', label='Benz')
```

上述调用 plt.plot( ) 时需同时设置 label，最后使用第 13 行的方式执行 legend( ) 的调用。其中，参数 loc 可以设置图例的位置，可以有下列设置方式。

```
'best':0,
'upper right':1
'upper left':2,
'lower left':3,
'lower right':4,
'right':5,        (与 'center right' 相同)
'center left':6,
'center right':7,
'lower center':8,
'upper center':9,
'center':10,
```

如果省略 loc 设置，则使用默认的 'best'，在应用时可以使用设置整数值，例如，设置 loc=0 与上述效果相同。若是考虑程序可读性，建议使用文字字符串方式设置，当然也可以直接设置数字。

程序实例 ch20_12_1.py：省略 loc 设置。

```
13  plt.legend()
```

执行结果 与 ch20_12.py 相同。

程序实例 ch20_12_2.py：设置 loc=0。

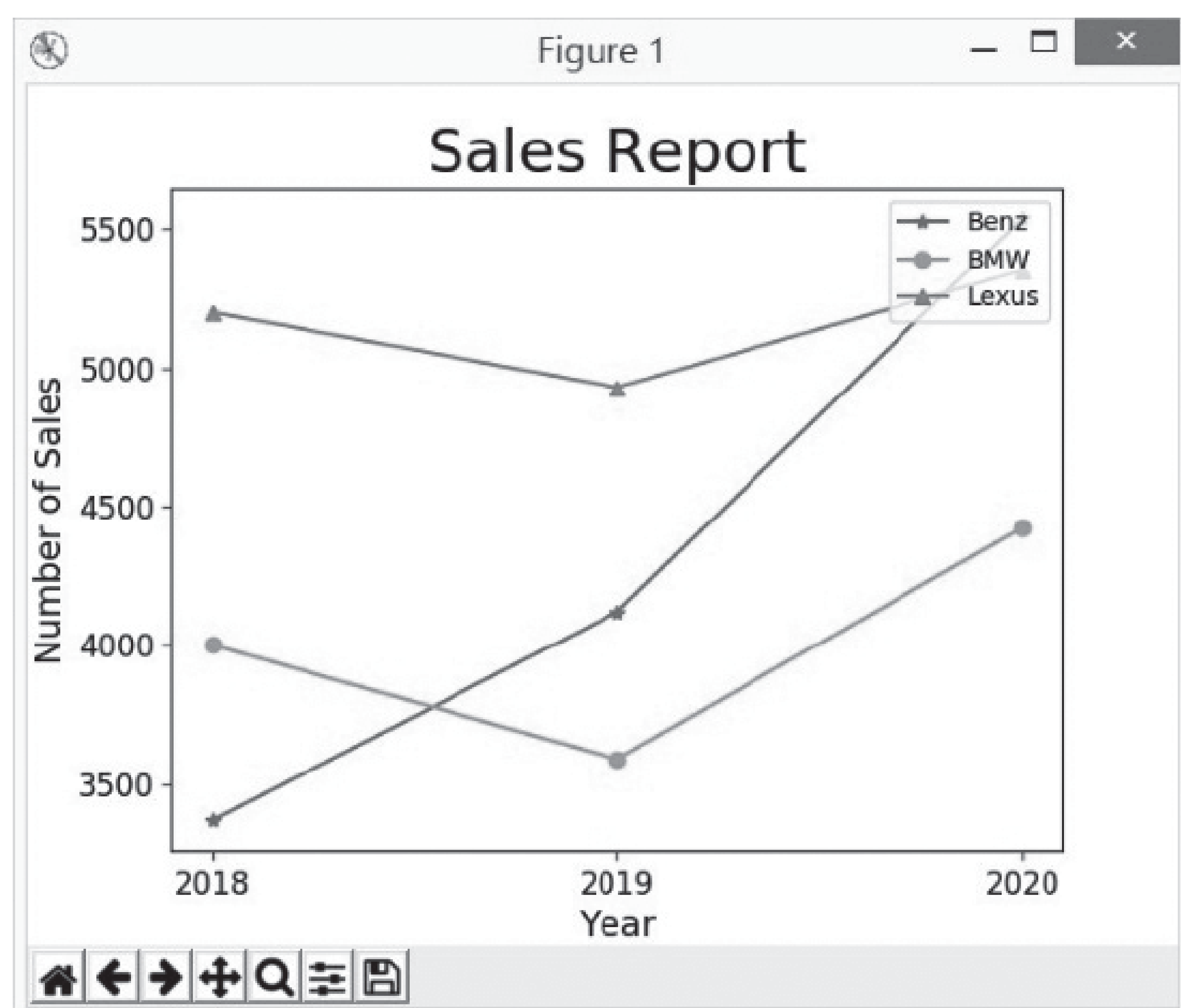
```
13  plt.legend(loc=0)
```

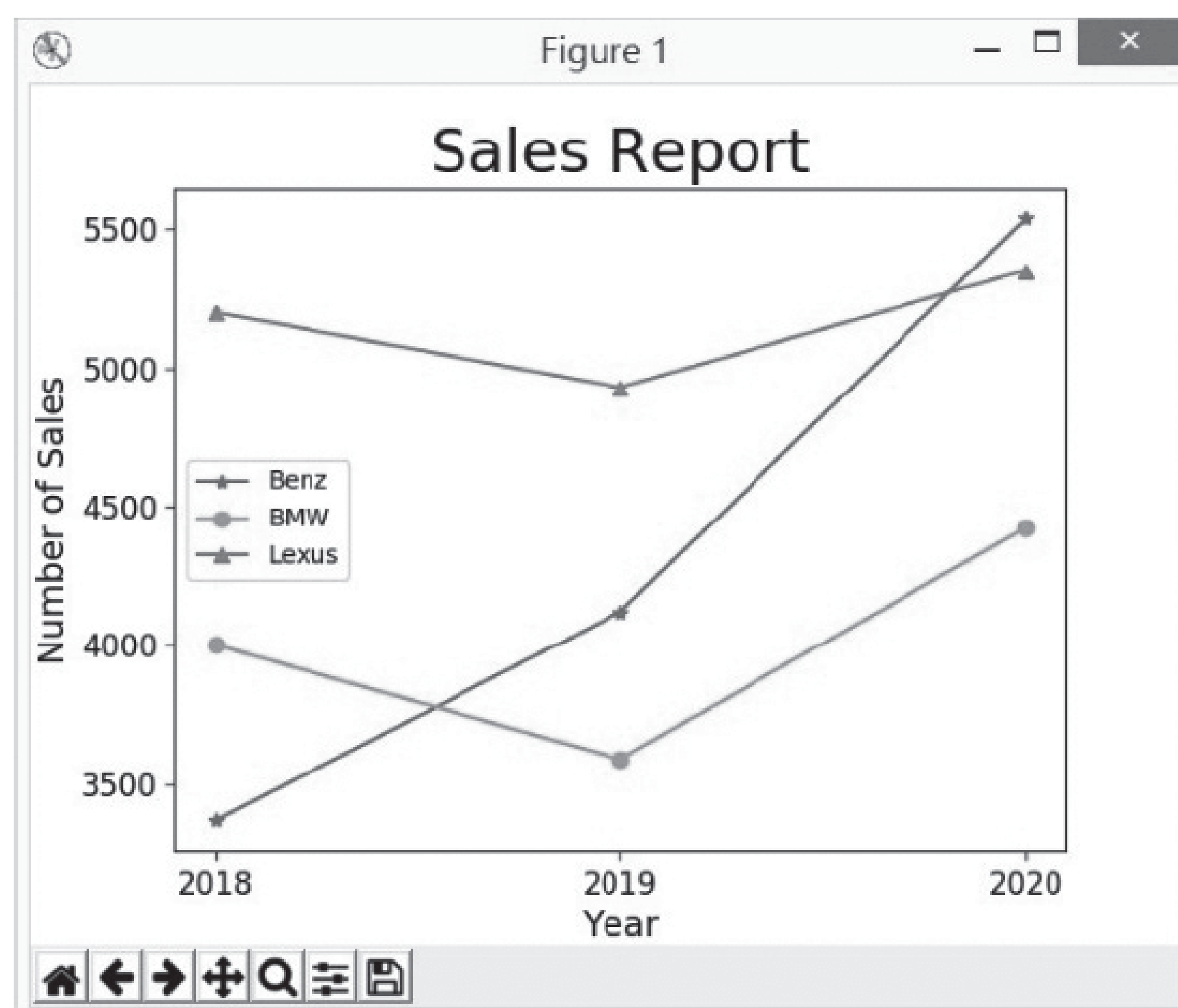
执行结果 与 ch20_12.py 相同。

程序实例 ch20_12_3.py：设置图例在右上角。

```
13  plt.legend(loc='upper right')
```

执行结果 下方左图。



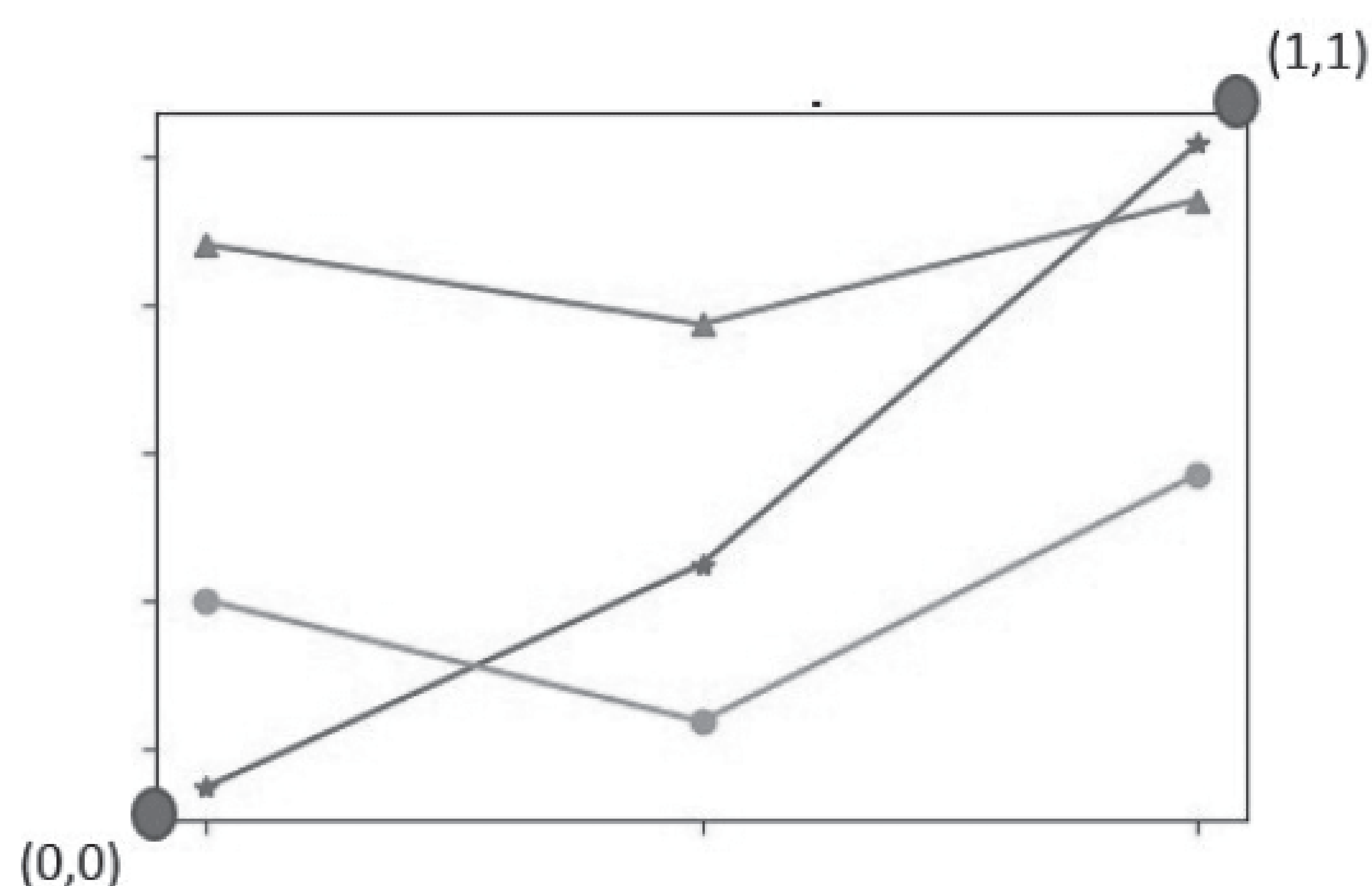


程序实例 ch20_12_4.py：设置图例在左边中央。

```
13  plt.legend(loc=6)
```

执行结果 上方右图。

经过上述解说，我们已经可以将图例放在图表内了。如果想将图例放在图表外，还要先了解坐标，在图表内左下角位置是（0,0），右上角是（1,1），如下所示。

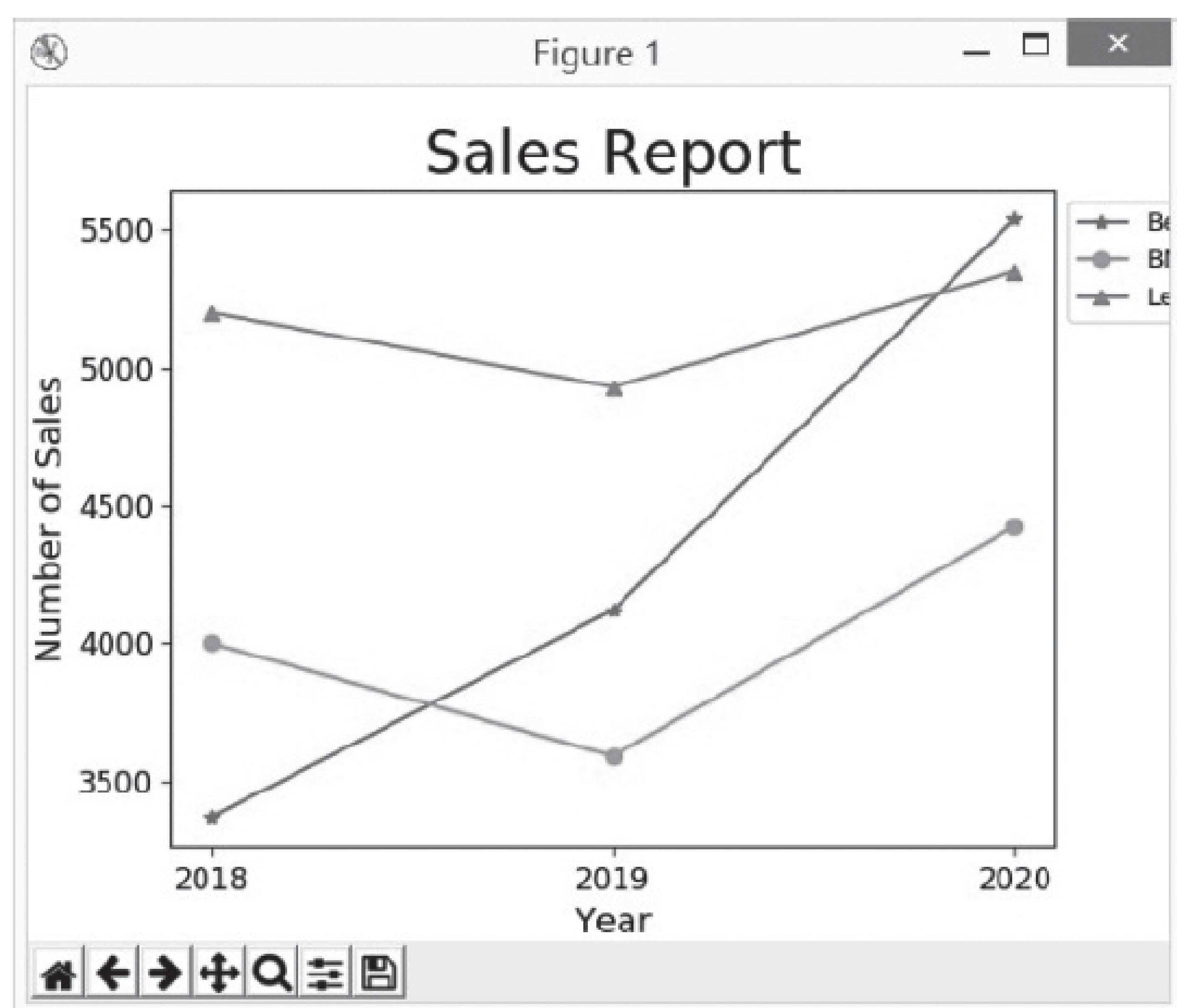


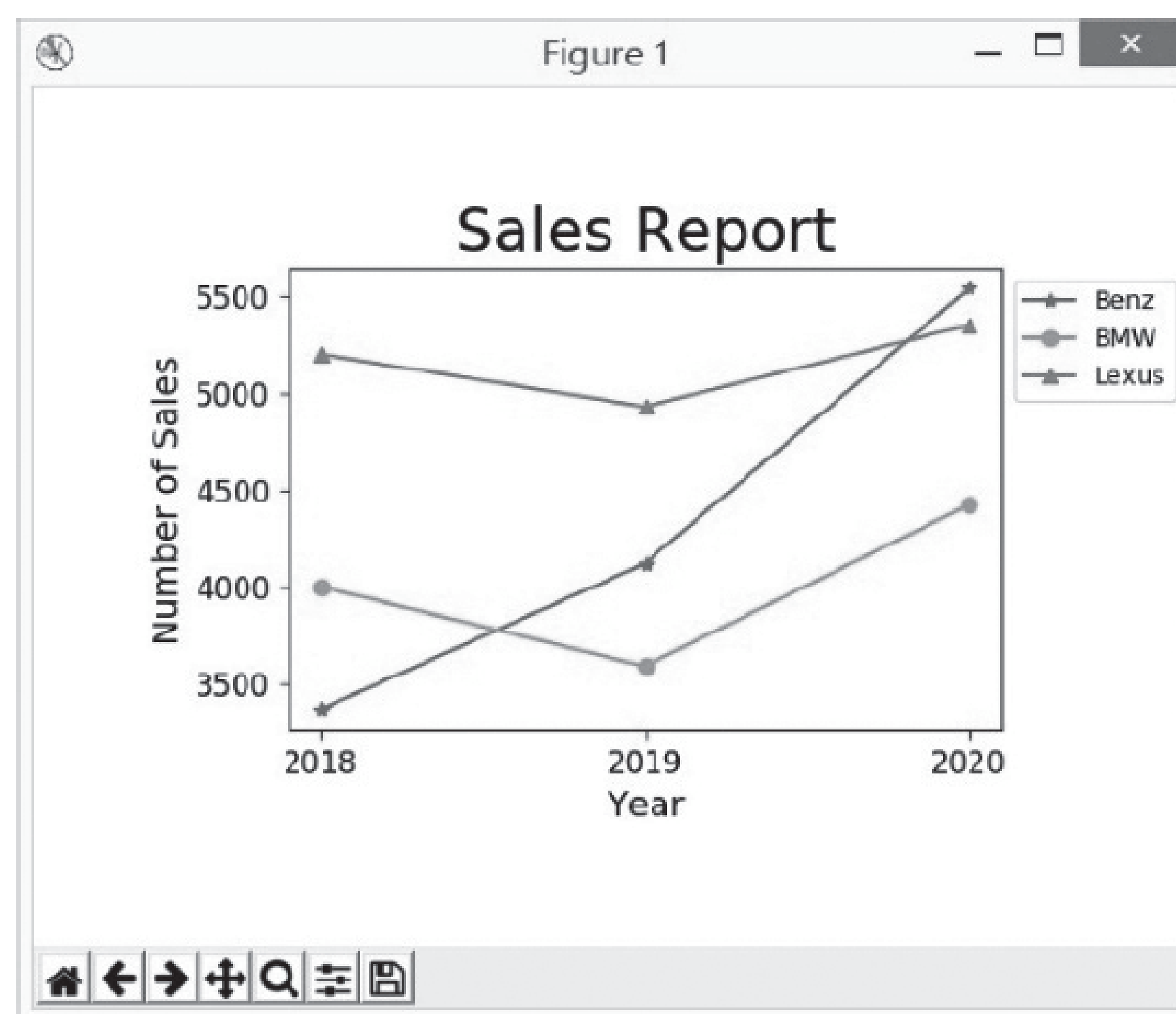
首先需使用 bbox_to_anchor( ) 当作 legend( ) 的一个参数，设置锚点（anchor），也就是图例位置，例如，如果想将图例放在图表右上角外侧，需设置 loc='upper left'，然后设置 bbox_to_anchor（1,1）。

程序实例 ch20_12_5.py：将图例放在图表右上角外侧。

```
13  plt.legend(loc=6, bbox_to_anchor=(1,1))
```

执行结果 下方左图。





上述最大的缺点是由于图表与 Figure 1 的留白不足，造成无法完整显示图例。matplotlib 模块内有 tight_layout( ) 函数，可利用 pad 参数，在图表与 Figure 1 间设置留白。

程序实例 ch20_12_6.py：设置 pad=7，重新设计 ch20_12_5.py。

```
13  plt.legend(loc='upper left',bbox_to_anchor=(1,1))
14  plt.tight_layout(pad=7)
```

执行结果 上方右图。

很明显右图改善了图例显示不完整的问题。如果将 pad 改为 h_pad/w_pad，可以分别设置高度 / 宽度的留白。

20-1-11　保存图表

图表设计完成后，可以使用 savefig( ) 保存图表，这个方法需放在 show( ) 的前方，表示先存储再显示图表。

程序实例 ch20_13.py：扩充 ch20_12.py，在屏幕显示图表前，先将图表存入目前文件夹的 out20_13.jpg。

```
1   # ch20_13.py
2   import matplotlib.pyplot as plt
3
4   Benz = [3367, 4120, 5539]                          # Benz线条
5   BMW = [4000, 3590, 4423]                           # BMW线条
6   Lexus = [5200, 4930, 5350]                         # Lexus线条
7
8   seq = [2018, 2019, 2020]                           # 年份
9   plt.xticks(seq)                                    # 设置x轴刻度
10  plt.plot(seq, Benz, '-*', label='Benz')
11  plt.plot(seq, BMW, '-o', label='BMW')
12  plt.plot(seq, Lexus, '-^', label='Lexus')
13  plt.legend(loc='best')
14  plt.title("Sales Report", fontsize=24)
15  plt.xlabel("Year", fontsize=14)
16  plt.ylabel("Number of Sales", fontsize=14)
17  plt.tick_params(axis='both', labelsize=12, color='red')
18  plt.savefig('out20_13.jpg', bbox_inches='tight')      #保存文件
19  plt.show()
```

执行结果 读者可以在 ch20 文件夹看到 out20_13.jpg 文件。

上述 plt.savefig( ) 第一个参数是所存的文件名，第二个参数代表将图表外多余的空间删除。

20-2 绘制散点图 scatter( )

除了使用 plot( ) 绘制散点图，本节将介绍另一种绘制散点图的常用方法 scatter( )。

20-2-1 基本散点图的绘制

绘制散点图可以使用 scatter( )，语法如下。更多参数应用未来几节会解说。

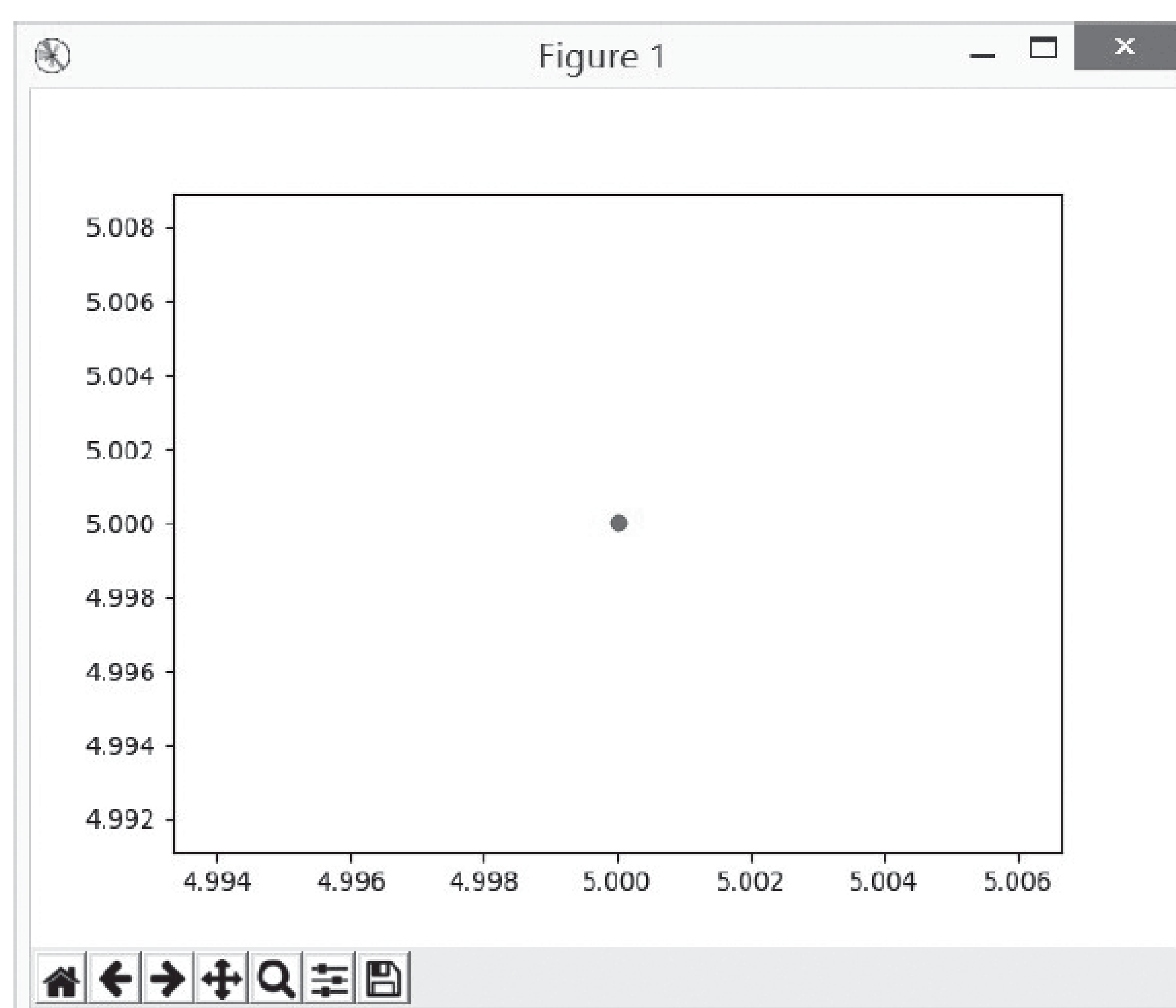
```
scatter(x, y, s, c)
```

上述语句相当于可以在（x,y）位置绘图，其中，（0,0）位置在左下角，x 轴刻度往右增加，y 轴刻度往上增加。s 是绘图点的大小，默认是 20。c 是颜色，默认是蓝色。暂时 s 与 c 都用默认值处理，未来将一步一步解说。

程序实例 ch20_14.py：在坐标轴 (5,5) 绘制一个点。

```
1  # ch20_14.py
2  import matplotlib.pyplot as plt
3
4  plt.scatter(5, 5)
5  plt.show()
```

执行结果



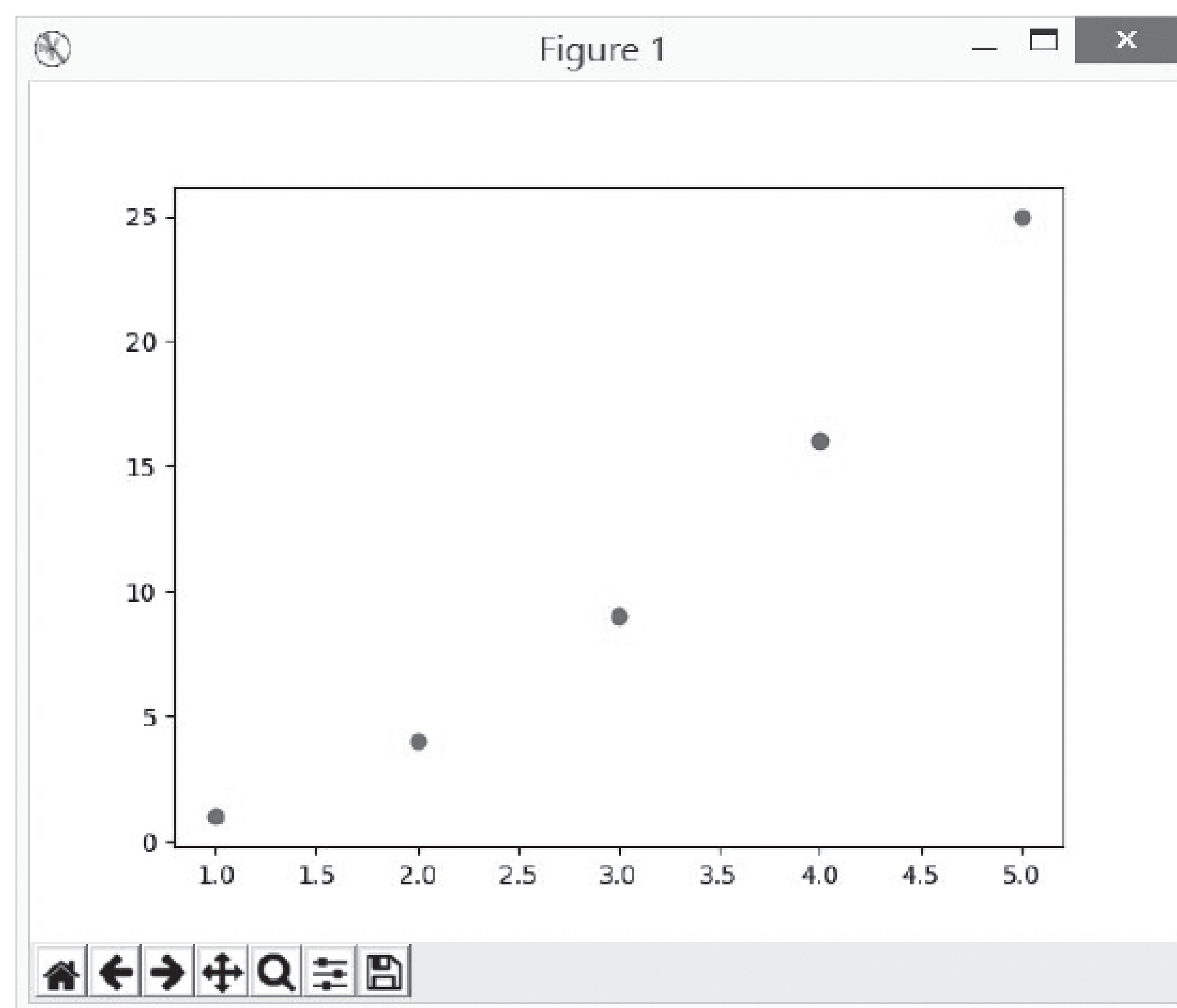
20-2-2 绘制系列点

如果想绘制一系列点，可以将这些点的 x 轴值放在一个列表中，y 轴值放在另一个列表中，然后将这两个列表当作参数放在 scatter( ) 中即可。

程序实例 ch20_15.py：绘制系列点的应用。

```
1  # ch20_15.py
2  import matplotlib.pyplot as plt
3
4  xpt = [1,2,3,4,5]
5  ypt = [1,4,9,16,25]
6  plt.scatter(xpt, ypt)
7  plt.show()
```

执行结果

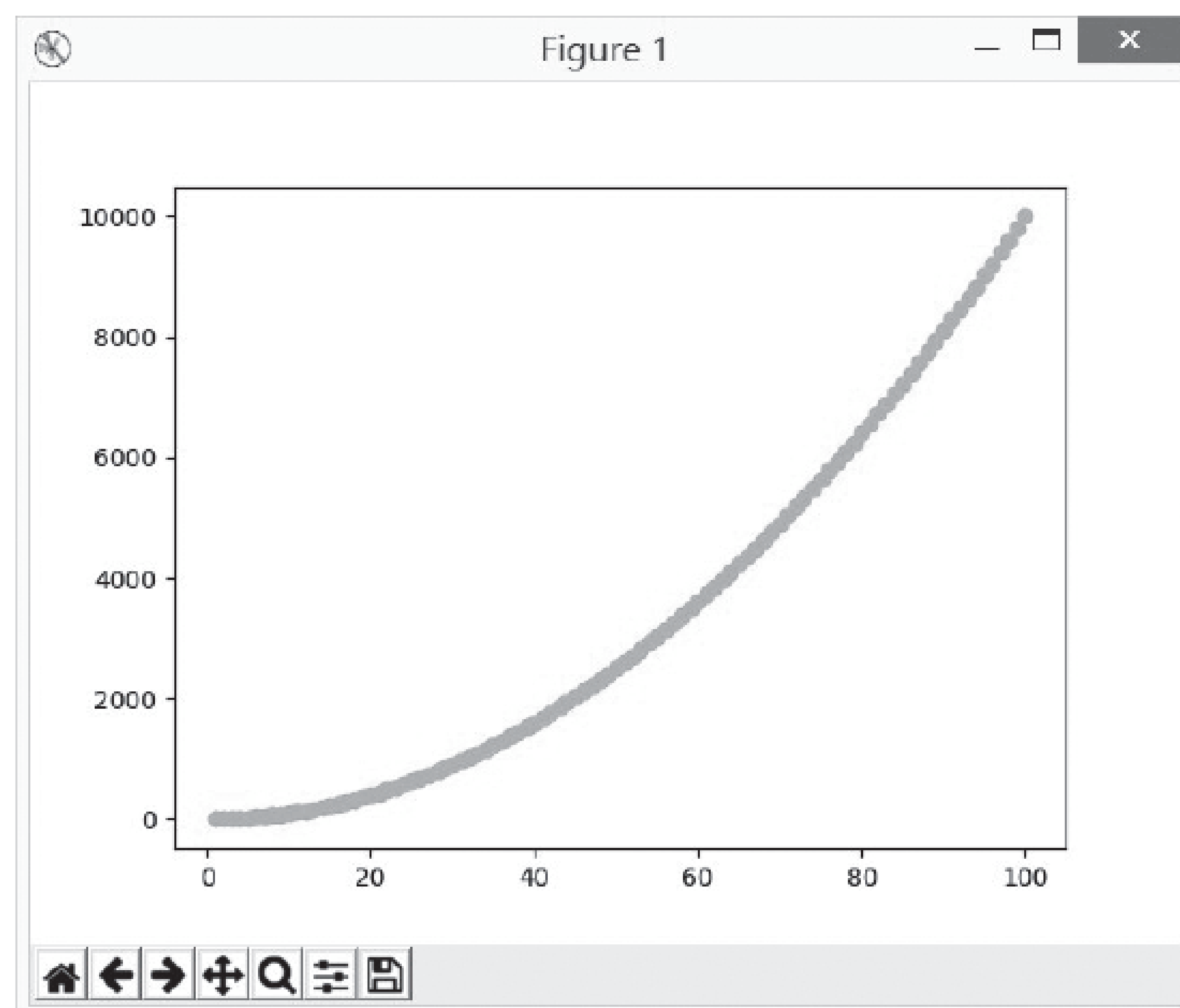


在程序设计时，有些系列点的坐标可能是由程序产生的，其实应用方式是一样的。另外，可以在 scatter( ) 内增加 color（也可用 c）参数，设置点的颜色。

程序实例 ch20_16.py：绘制一系列黄色的点，这个系列中有 100 个点，x 轴的点由 range(1,101) 产生，相对应 y 轴的值则是 x 的平方值。

```
1  # ch20_16.py
2  import matplotlib.pyplot as plt
3
4  xpt = list(range(1,101))          # 建立1~100序列x坐标点
5  ypt = [x**2 for x in xpt]         # 以x平方方式建立y坐标点
6  plt.scatter(xpt, ypt, color='y')
7  plt.show()
```

执行结果



上述程序第 6 行使用直接的指定色彩，也可以使用 RGB（Red, Green, Blue）颜色模式设置色彩，RGB( ) 内每个参数数值为 0 ～ 1。

20-2-3　设置绘图区间

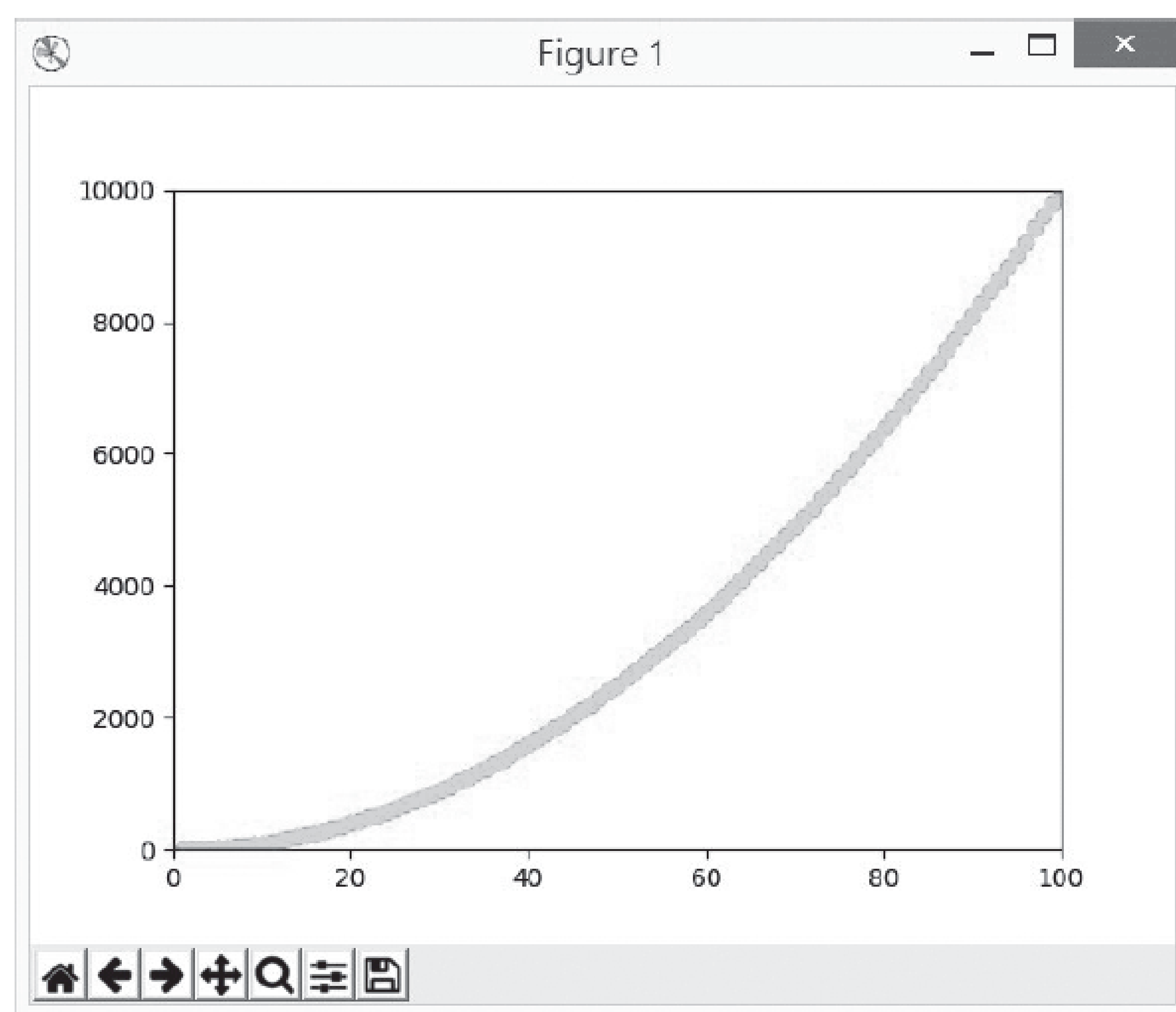
可以使用 axis( ) 设置绘图区间，语法格式如下。

```
axis([xmin, xmax, ymin, ymax])        # 分别代表 x 轴和 y 轴的最小和最大区间
```

程序实例 ch20_17.py：设置绘图区间为 [0,100,0,10000] 的应用，读者可以将这个执行结果与 ch20_16.py 做比较。另外，第 7 行以不同方式建立色彩。

```
1  # ch20_17.py
2  import matplotlib.pyplot as plt
3
4  xpt = list(range(1,101))              # 建立1~100序列x坐标点
5  ypt = [x**2 for x in xpt]             # 以x平方方式建立y坐标点
6  plt.axis([0, 100, 0, 10000])          # 留意参数是列表
7  plt.scatter(xpt, ypt, c=(0, 1, 0))    # 绿色
8  plt.show()
```

执行结果



上述程序第 5 行是依据 xpt 列表产生 ypt 列表值的方式。由于网络上的文章大多使用数组方式产生图表，所以 20-3 节将对此进行说明，期待为读者建立基础。

20-3　Numpy 模块

Numpy 是 Python 的一个扩充模块，主要是可以支持多维度空间的数组与矩阵运算，本节将使用其最简单的产生数组的功能做解说，由此可以将这个功能扩充到数据图表的设计。程序中 Numpy 模块的第一个字母 n 是小写，使用前需导入 Numpy 模块，如下所示。

```
import numpy as np
```

第 23 章将对 Numpy 模块做更多说明。

20-3-1 建立一个简单的数组 linspace( ) 和 arange( )

在 Numpy 模块中最基本的就是 linspace( ) 方法，使用它可以很方便地产生等距的数组，语法如下。

```
linspace(start, end, num)               # 这是最常用的简化语法
```

start 是起始值，end 是结束值，num 是设置产生多少个等距点的数组值，num 的默认值是 50。

在网络上阅读他人使用 Python 设计的图表时，另一个常见的产生数组的方法是 arange( )，语法如下。

```
arange(start, stop, step)               # start 和 step 可以省略
```

start 是起始值，如果省略默认值是 0；stop 是结束值，但是所产生的数组不包含此值；step 是数组相邻元素的间距，如果省略默认值是 1。

程序实例 ch20_18.py：建立 0 ～ 10 的数组。

```
1  # ch20_18.py
2  import numpy as np
3
4  x1 = np.linspace(0, 10, num=11)       # 使用linspace()产生数组
5  print(type(x1), x1)
6  x2 = np.arange(0,11,1)                # 使用arange()产生数组
7  print(type(x2), x2)
8  x3 = np.arange(11)                    # 简化语法产生数组
9  print(type(x3), x3)
```

执行结果

```
===================== RESTART: D:\Python\ch20\ch20_18.py =====================
<class 'numpy.ndarray'> [ 0.   1.   2.   3.   4.   5.   6.   7.   8.   9.  10.]
<class 'numpy.ndarray'> [ 0  1  2  3  4  5  6  7  8  9 10]
<class 'numpy.ndarray'> [ 0  1  2  3  4  5  6  7  8  9 10]
>>>
```

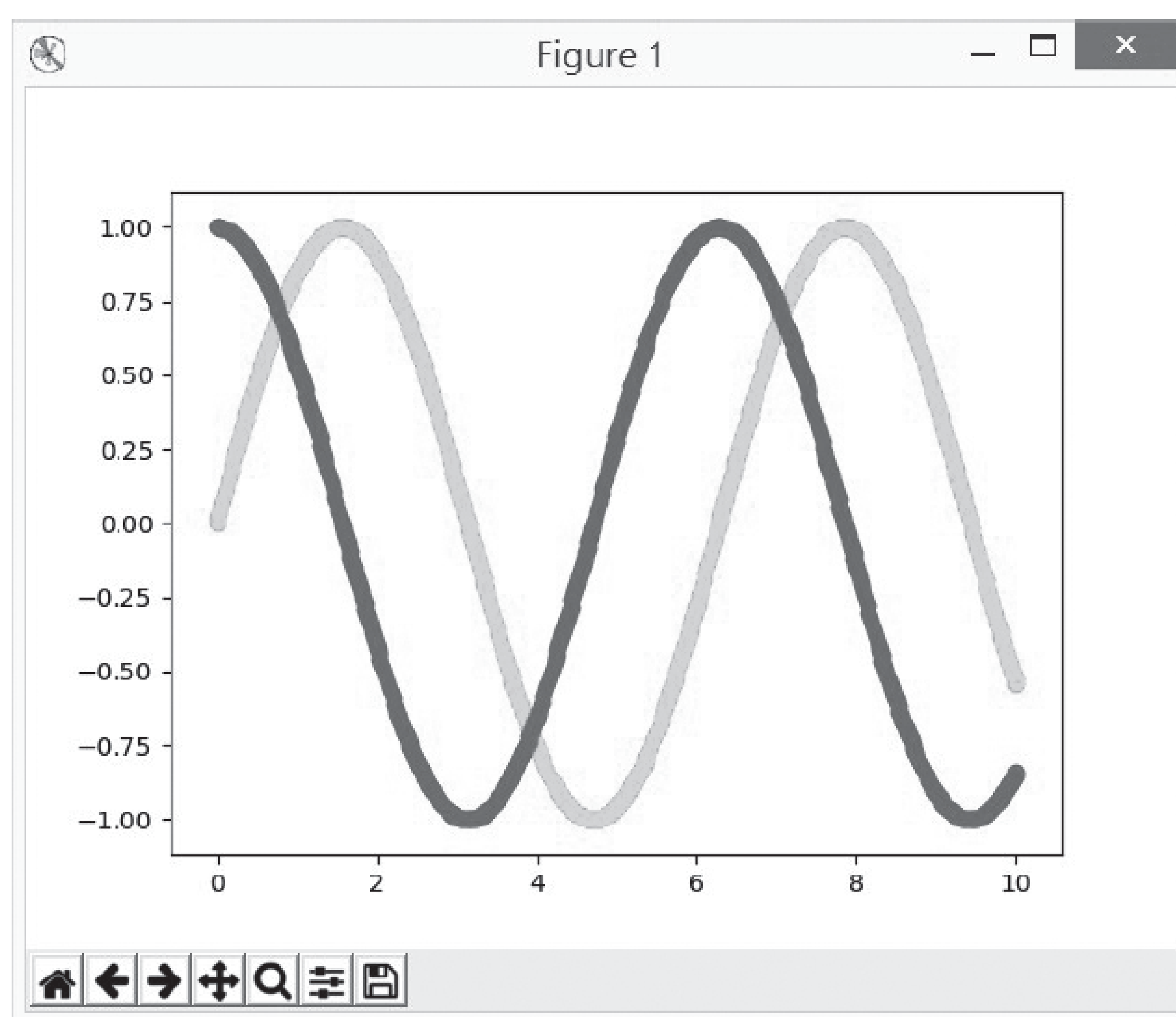
20-3-2 绘制波形

在中学数学中我们有学过 sin( ) 和 cos( ) 的概念，其实有了数组数据，可以很方便地绘制 sin 和 cos 的波形变化。

程序实例 ch20_19.py：绘制 sin( ) 和 cos( ) 的波形，在这个实例中调用 plt.scatter( ) 方法两次，相当于也可以绘制两次波形图表。

```
 1  # ch20_19.py
 2  import matplotlib.pyplot as plt
 3  import numpy as np
 4
 5  xpt = np.linspace(0, 10, 500)              # 建立含500个元素的数组
 6  ypt1 = np.sin(xpt)                         # y数组的变化
 7  ypt2 = np.cos(xpt)
 8  plt.scatter(xpt, ypt1, color=(0, 1, 0)) # 绿色
 9  plt.scatter(xpt, ypt2)                     # 默认颜色
10  plt.show()
```

执行结果

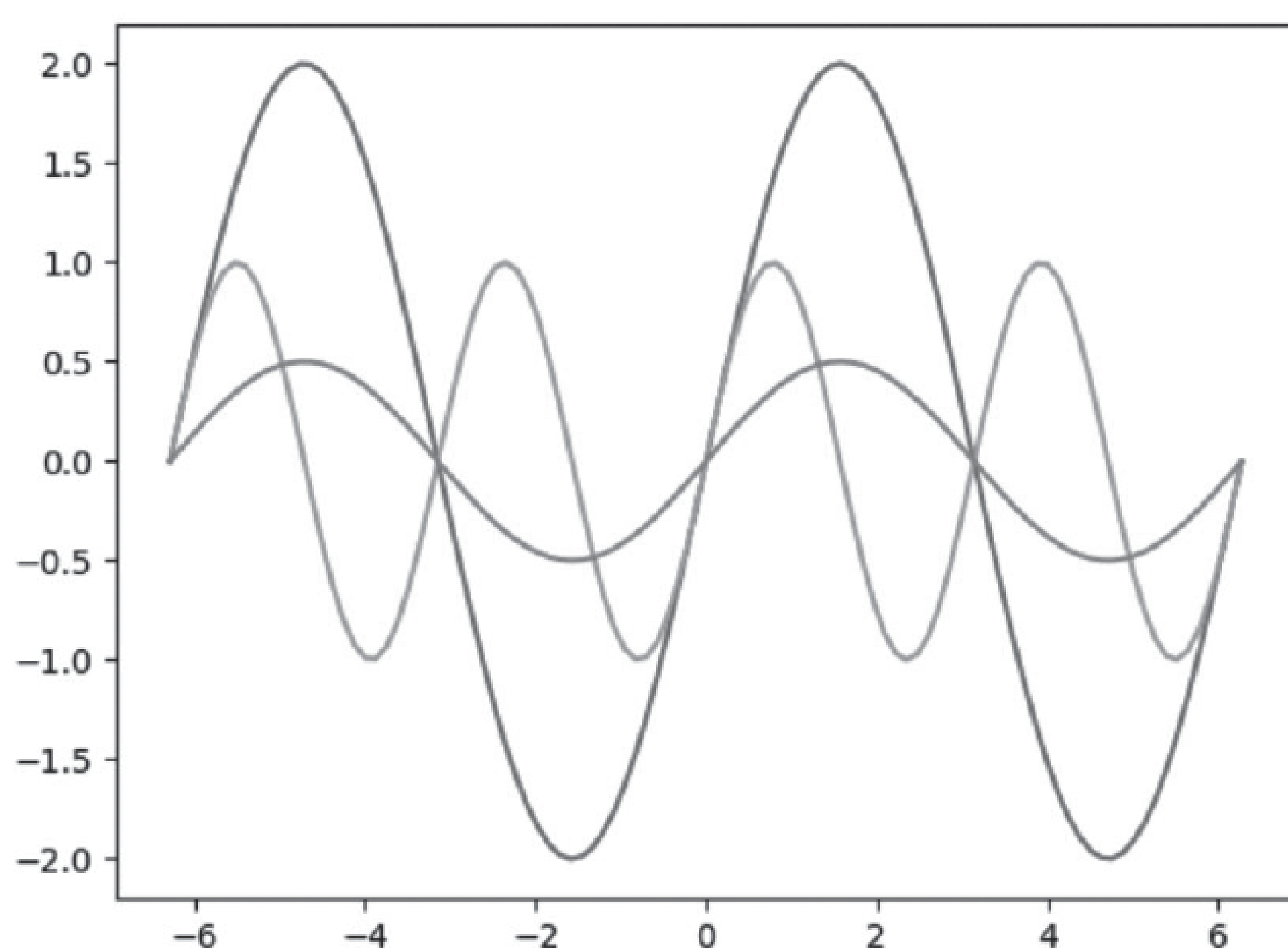


其实一般在绘制波形时，最常用的还是 plot( ) 方法。

程序实例 ch20_19_1.py：使用系统默认颜色，绘制不同波形。

```
1  # ch20_19_1.py
2  import matplotlib.pyplot as plt
3  import numpy as np
4
5  left = -2 * np.pi
6  right = 2 * np.pi
7  x = np.linspace(left, right, 100)
8
9  f1 = 2 * np.sin(x)                    # y数组的变化
10 f2 = np.sin(2*x)
11 f3 = 0.5 * np.sin(x)
12
13 plt.plot(x, f1)
14 plt.plot(x, f2)
15 plt.plot(x, f3)
16 plt.show()
```

执行结果



20-3-3　建立宽度不等的散点图

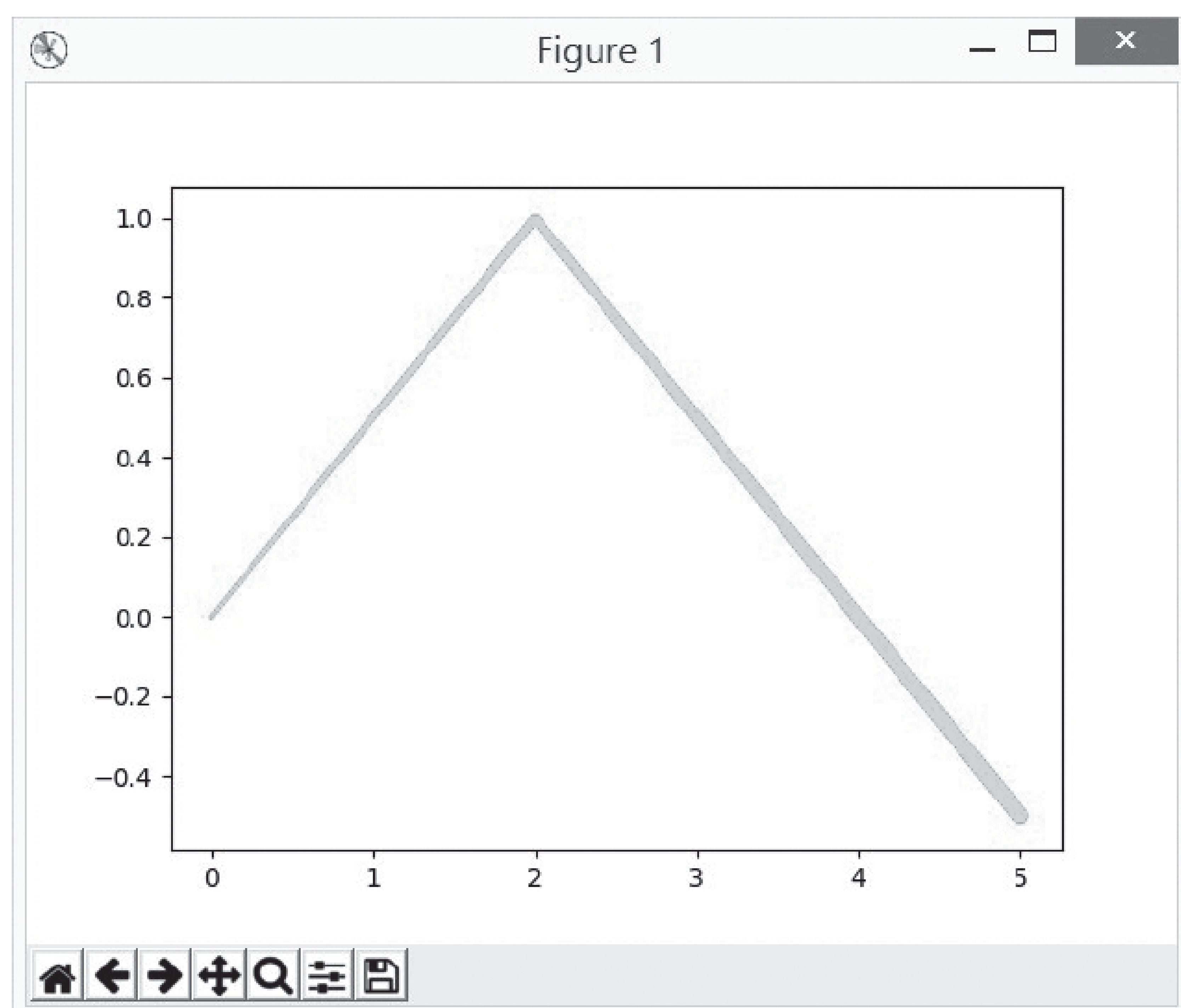
在 scatter( ) 方法中，（x,y）数据可以是列表也可以是矩阵，默认所绘制点大小 s 的值是 20，这个 s 可以是一个值也可以是一个数组数据，当它是一个数组数据时，利用更改数组值的大小，就可以建立不同大小的散点图。

在使用 Python 绘制散点图时，如果在两个点之间绘制了上百或上千个点，则可以产生绘制线条的视觉，如果再加上每个点的大小是不同的，且依一定规律变化，则可以有特殊效果。

程序实例 ch20_20.py：建立一个不等宽度的图形。

```
1  # ch20_20.py
2  import matplotlib.pyplot as plt
3  import numpy as np
4
5  xpt = np.linspace(0, 5, 500)                          # 建立含500个元素的数组
6  ypt = 1 - 0.5*np.abs(xpt-2)                           # y数组的变化
7  lwidths = (1+xpt)**2                                  # 宽度数组
8  plt.scatter(xpt, ypt, s=lwidths, color=(0, 1, 0))     # 绿色
9  plt.show()
```

执行结果



20-3-4　填满区间

在绘制波形时，有时候想要填满区间，此时可以使用 matplotlib 模块的 fill_between( ) 方法，基本语法如下。

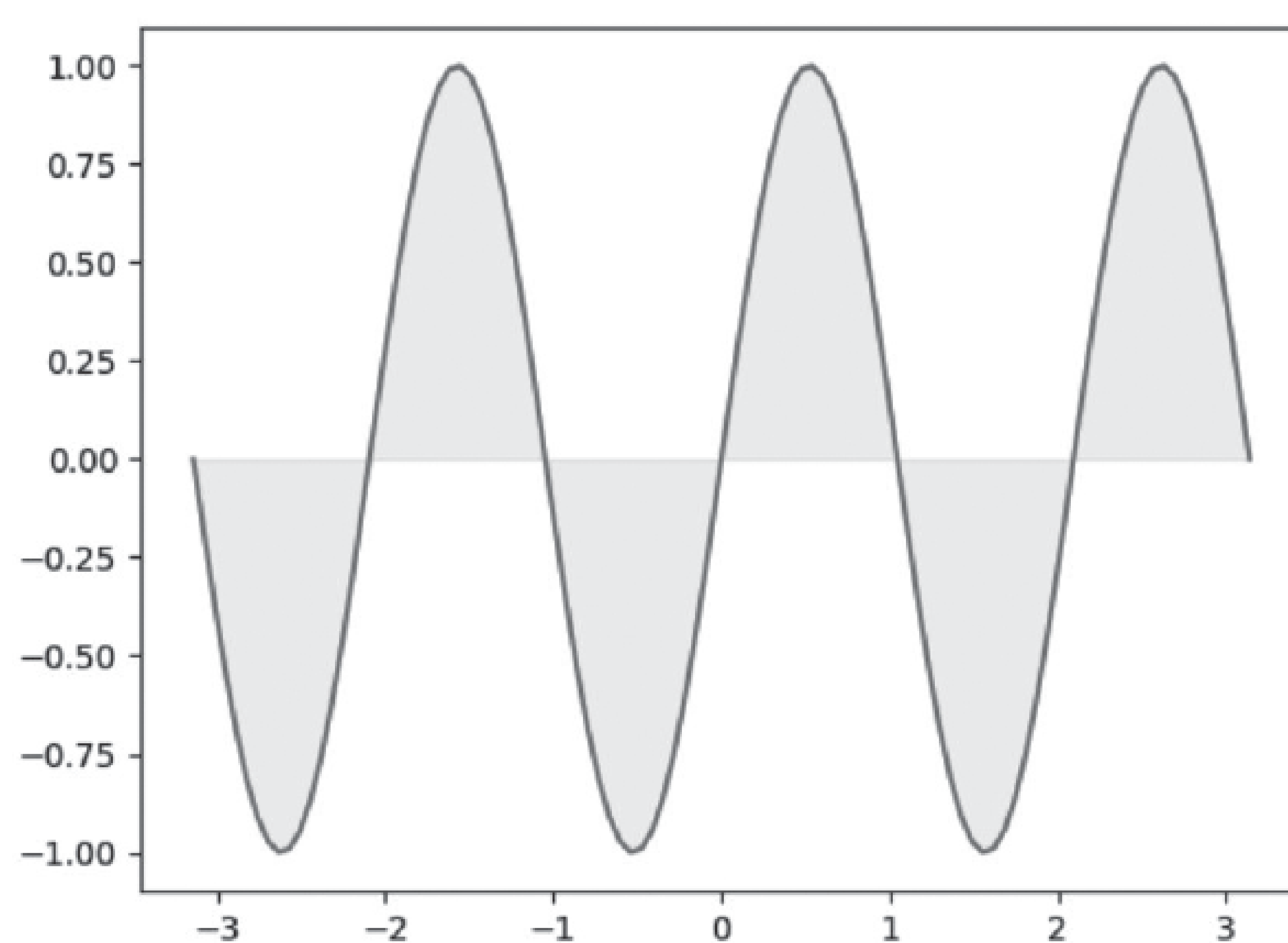
fill_between(x, y1, y2, color, alpha, options, …)　# options 是其他参数

上述语句会填满所有相对 x 轴数列 y1 和 y2 的区间，如果不指定填满颜色会使用默认的线条颜色填满，通常填满颜色会用较淡的颜色，所以可以设置 alpha 参数将颜色调淡。

程序实例 ch20_20_1.py：填满区间 0 ～ y，所使用的 y 轴值函数式为 sin(3x)。

```
1  # ch20_20_1.py
2  import matplotlib.pyplot as plt
3  import numpy as np
4
5  left = -np.pi
6  right = np.pi
7  x = np.linspace(left, right, 100)
8  y = np.sin(3*x)                    # y数组的变化
9
10 plt.plot(x, y)
11 plt.fill_between(x, 0, y, color='green', alpha=0.1)
12 plt.show()
```

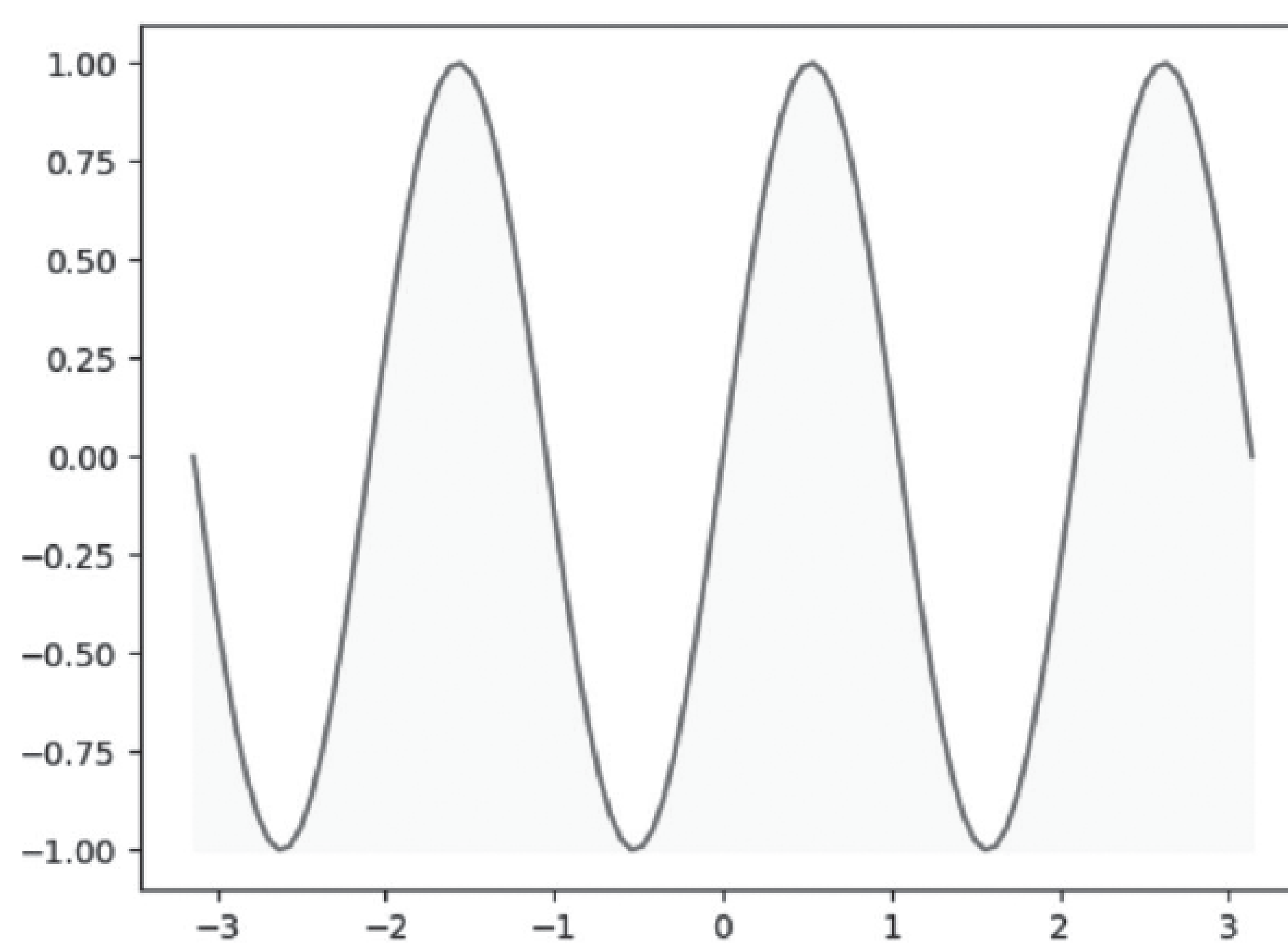
执行结果



程序实例 ch20_20_2.py：填满区间 1 ～ y，所使用的 y 轴值函数式为 sin(3x)。

```
1  # ch20_20_2.py
2  import matplotlib.pyplot as plt
3  import numpy as np
4
5  left = -np.pi
6  right = np.pi
7  x = np.linspace(left, right, 100)
8  y = np.sin(3*x)                    # y数组的变化
9
10 plt.plot(x, y)
11 plt.fill_between(x, -1, y, color='yellow', alpha=0.3)
12 plt.show()
```

执行结果



23-3-5 色彩映射

至今我们针对一组数组或列表所绘制的图都是单色的，若是以 ch20_20.py 第 8 行为例，色彩设置是 color=（0,1,0），这是固定颜色的用法。在色彩的使用中允许色彩随着数据而做变化，此时色彩的变化是根据所设置的**色彩映射值**（color mapping）而定。例如，有一个**色彩映射值**是 rainbow，内容如下。

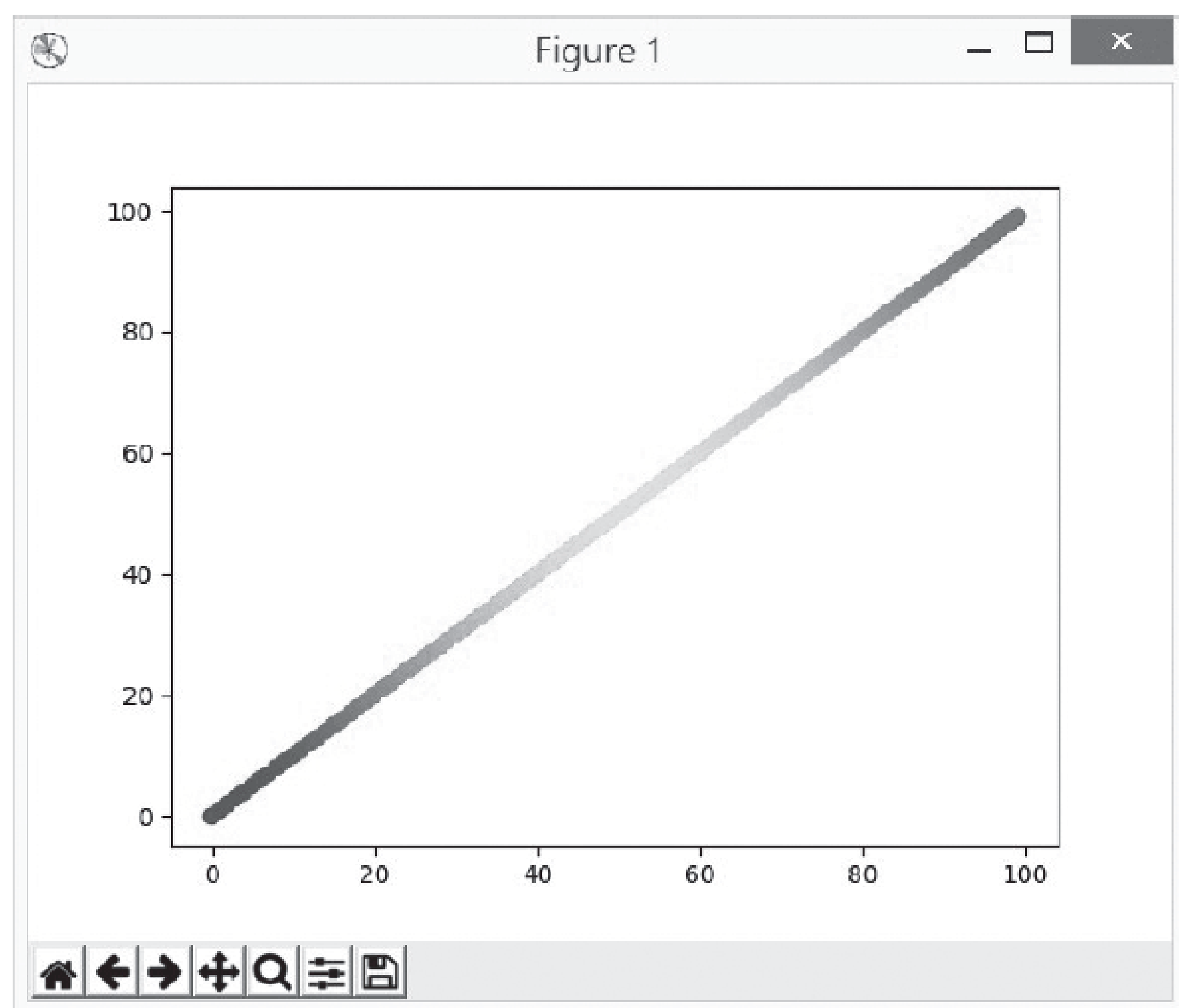


在数组或列表中，数值低的颜色在左边，会随着数值变高往右边移动。当然在程序设计中，需在 scatter( ) 中增加 color（也可用 c）设置，这时 color 的值就变成一个数组或列表。然后需增加参数 cmap（英文是 color map），这个参数主要是指定使用哪一种**色彩映射值**。

程序实例 ch20_20_3.py：色彩映射的应用。

```
1  # ch20_20_3.py
2  import matplotlib.pyplot as plt
3  import numpy as np
4
5  x = np.arange(100)
6  y = x
7  t = x
8  plt.scatter(x, y, c=t, cmap='rainbow')
9  plt.show()
```

执行结果

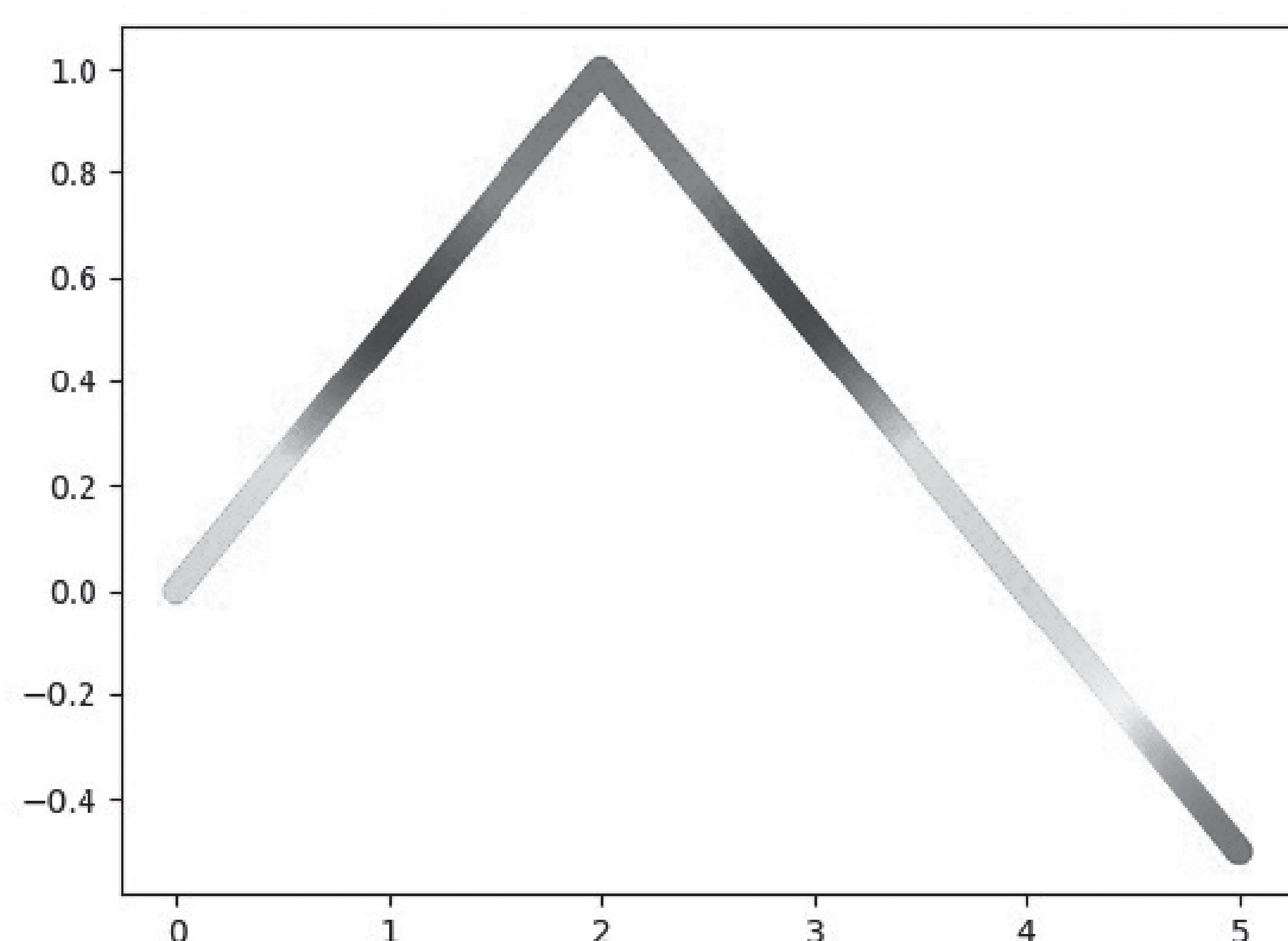


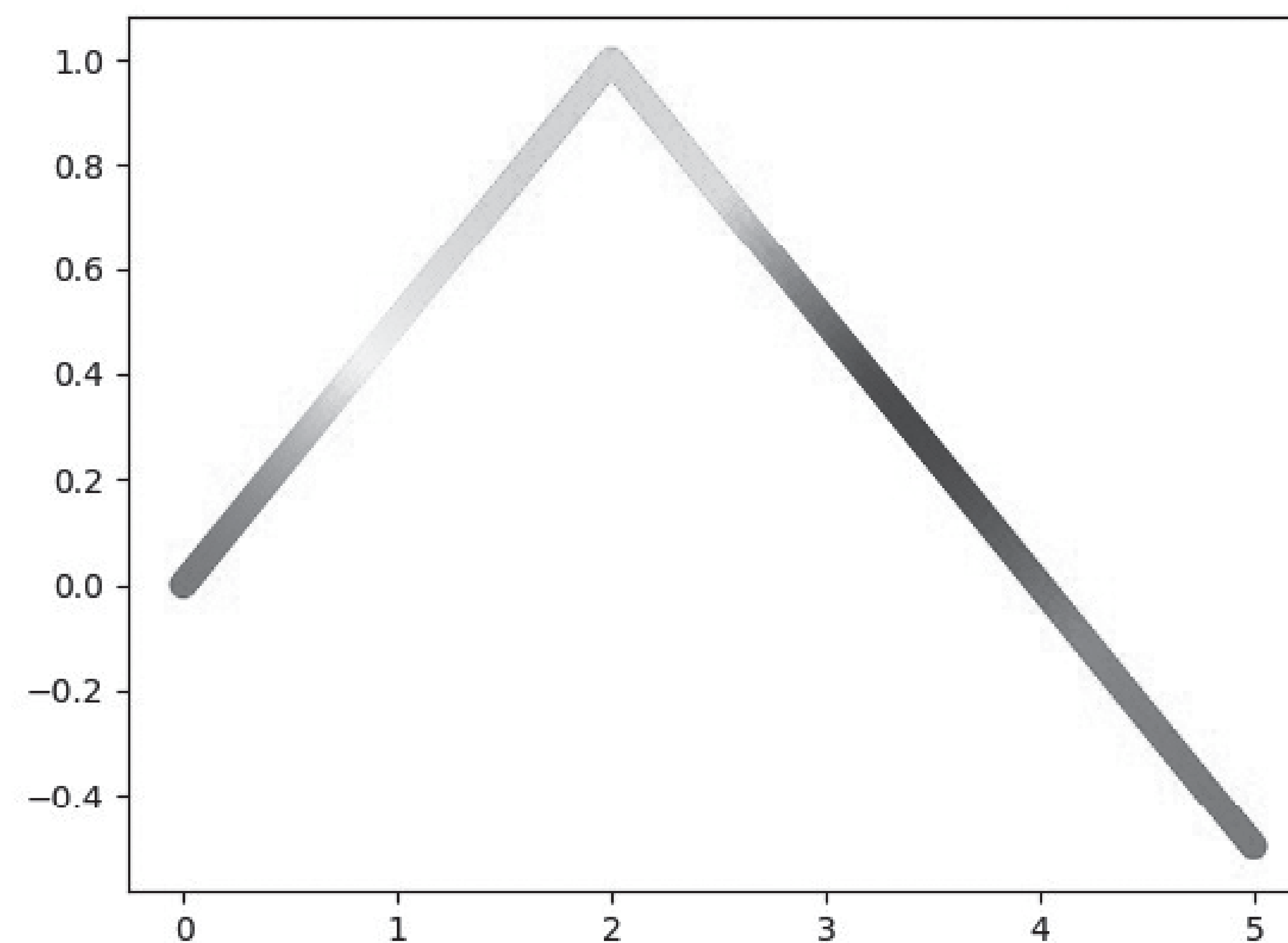
有时候在程序设计时，色彩映射也可以设置为根据 x 轴的值做变化，或是根据 y 轴的值做变化，整个效果是不一样的。

程序实例 ch20_20_4.py：重新设计 ch20_20.py，主要是设置固定点的宽度为 50，将色彩改为依 y 轴值变化，同时使用 hsv 色彩映射表。

```
8  plt.scatter(xpt, ypt, s=50, c=ypt, cmap='hsv')          # 色彩随y轴值变化
```

执行结果 如下方左图所示。





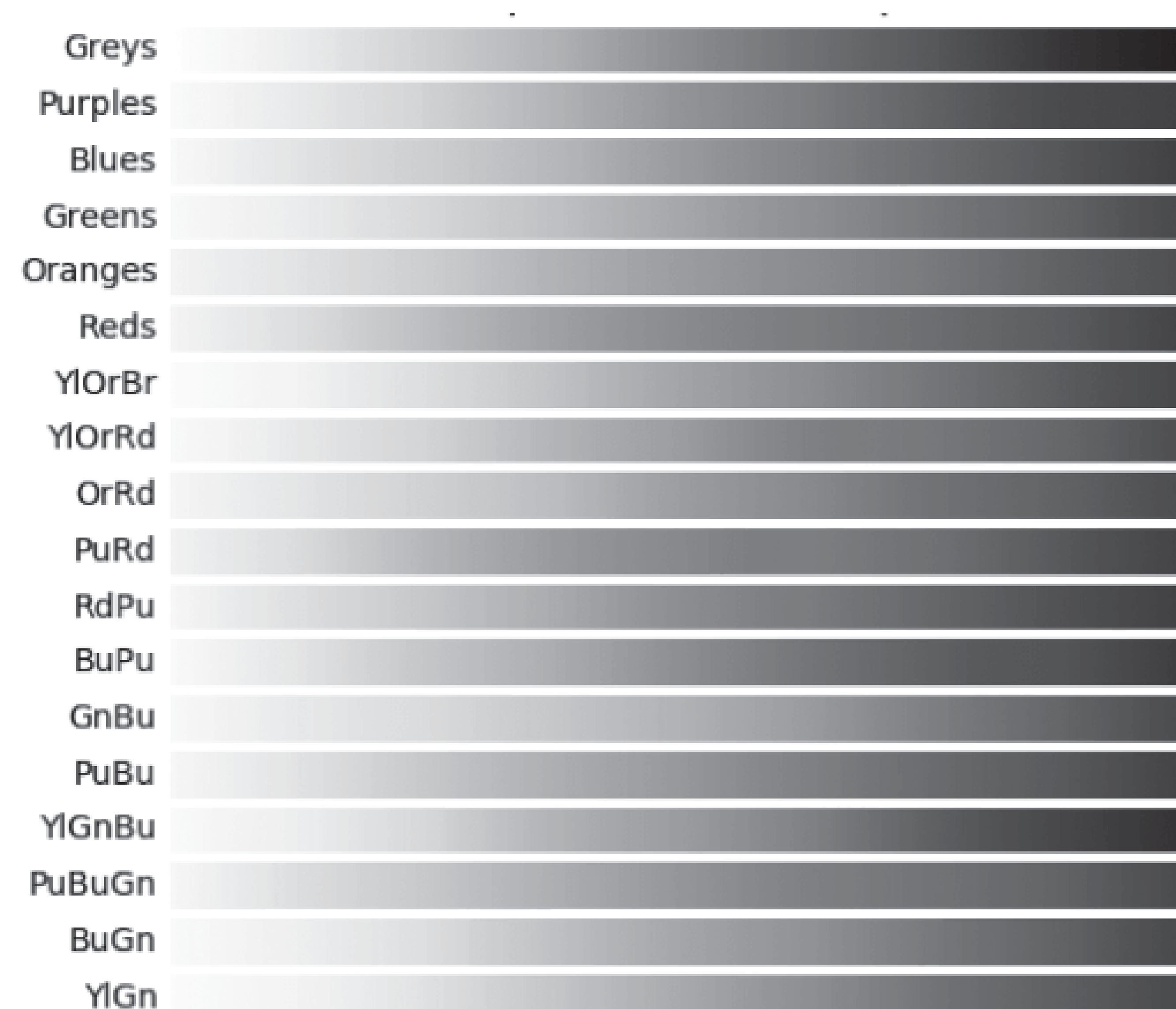
程序实例 ch20_20_5.py：重新设计 ch20_20_4.py，主要是将色彩改为依 x 轴值变化。

```
8  plt.scatter(xpt, ypt, s=50, c=xpt, cmap='hsv')          # 色彩随x轴值变化
```

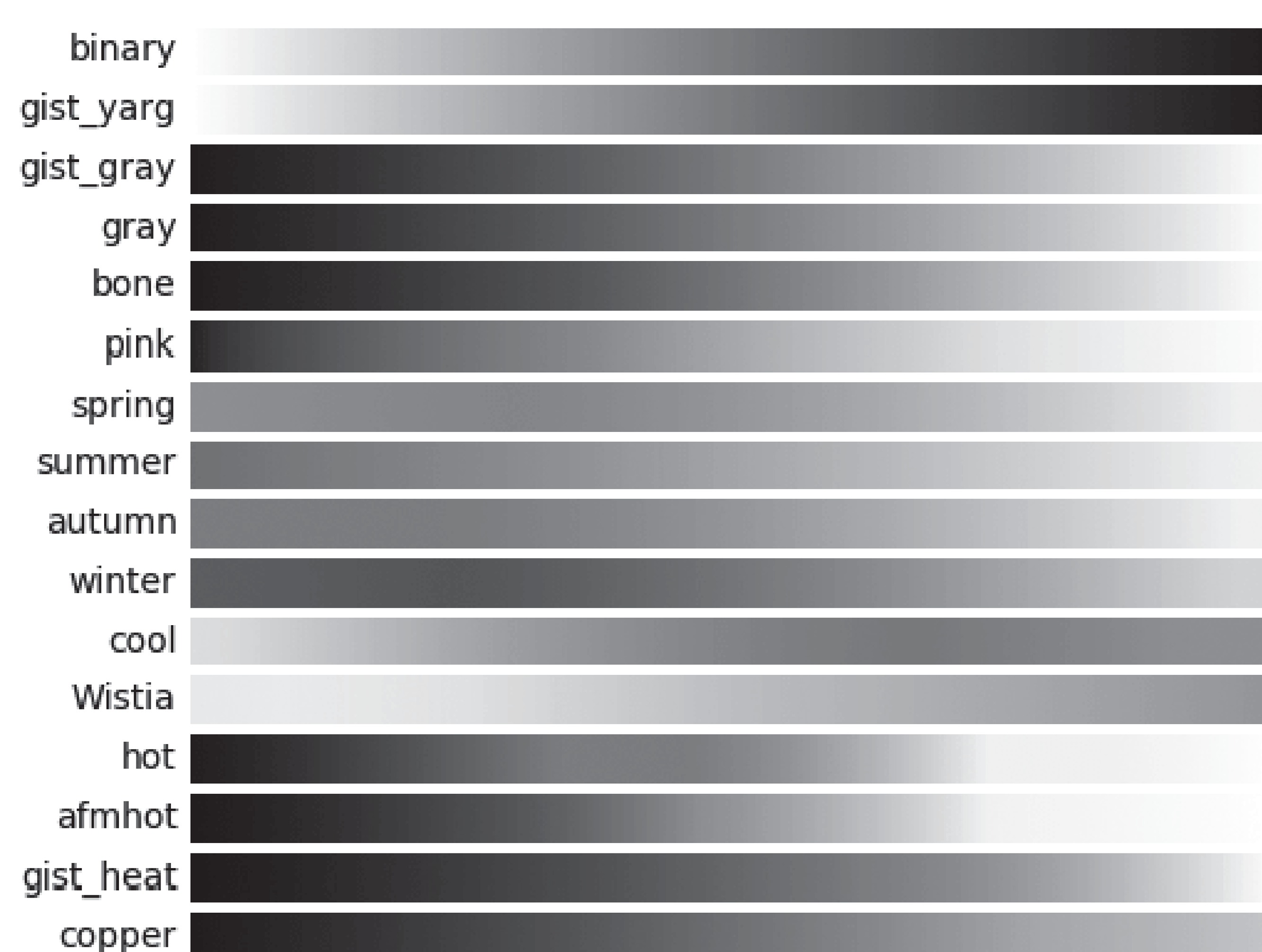
执行结果 如上方右图所示。

目前，matplotlib 协会所提供的色彩映射内容如下。

（1）序列色彩映射表。



（2）序列 2 色彩映射表。



（3）直觉一致的色彩映射表。



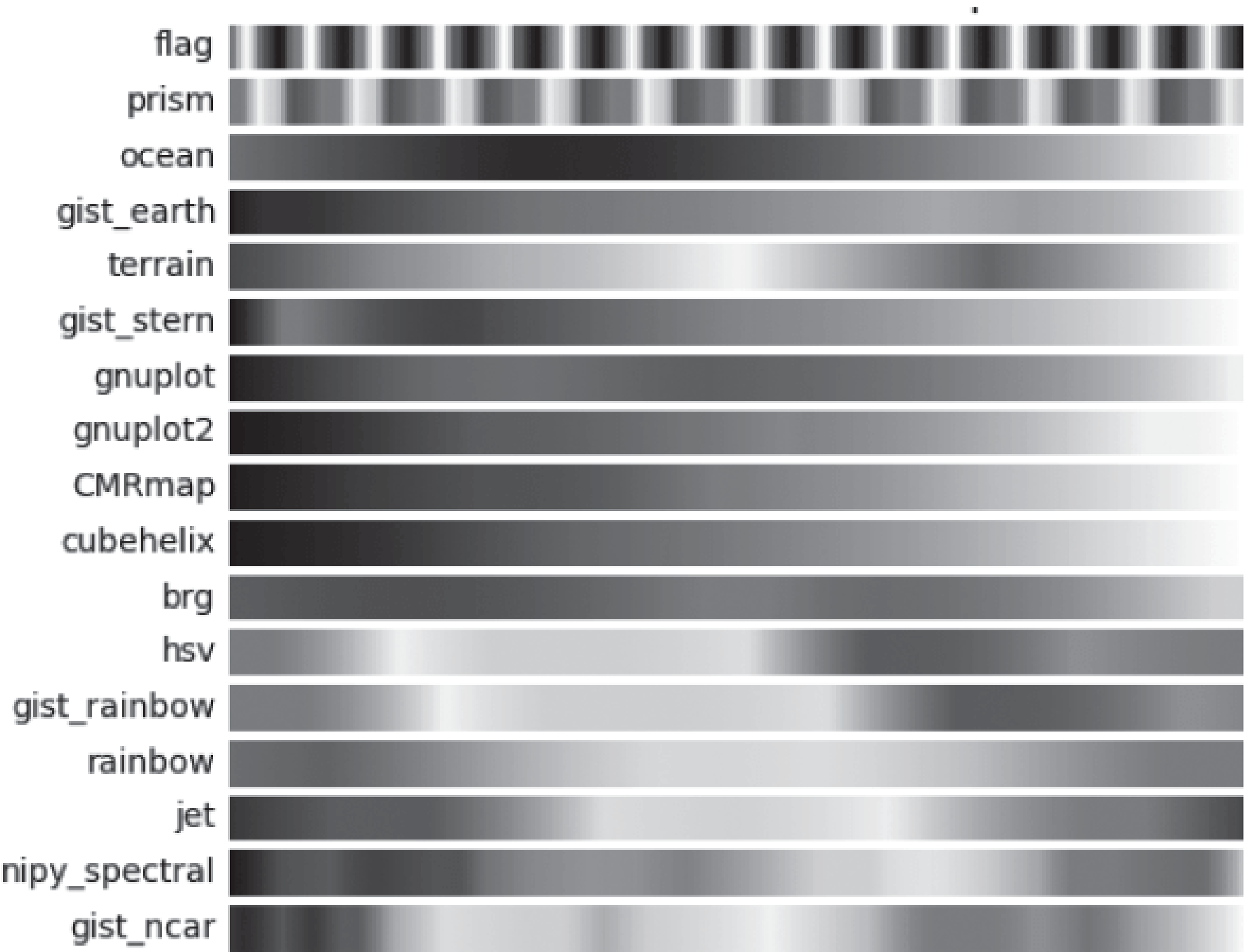
（4）发散式的色彩映射表。



（5）定性色彩映射表。



（6）杂项色彩映像表。



数据源 matplotlib 协会

http://matplotlib.org/examples/color/colormaps_reference.html

将来读者做大数据研究时，当收集了大量的数据后，可以将数据以图表显示，然后用色彩变化判断整个数据趋势。

20-4 随机数的应用

随机数在统计的应用中是非常重要的知识，本节试着用随机数方法，介绍 Python 的随机数分布，这一节将介绍下列随机数生成方法。

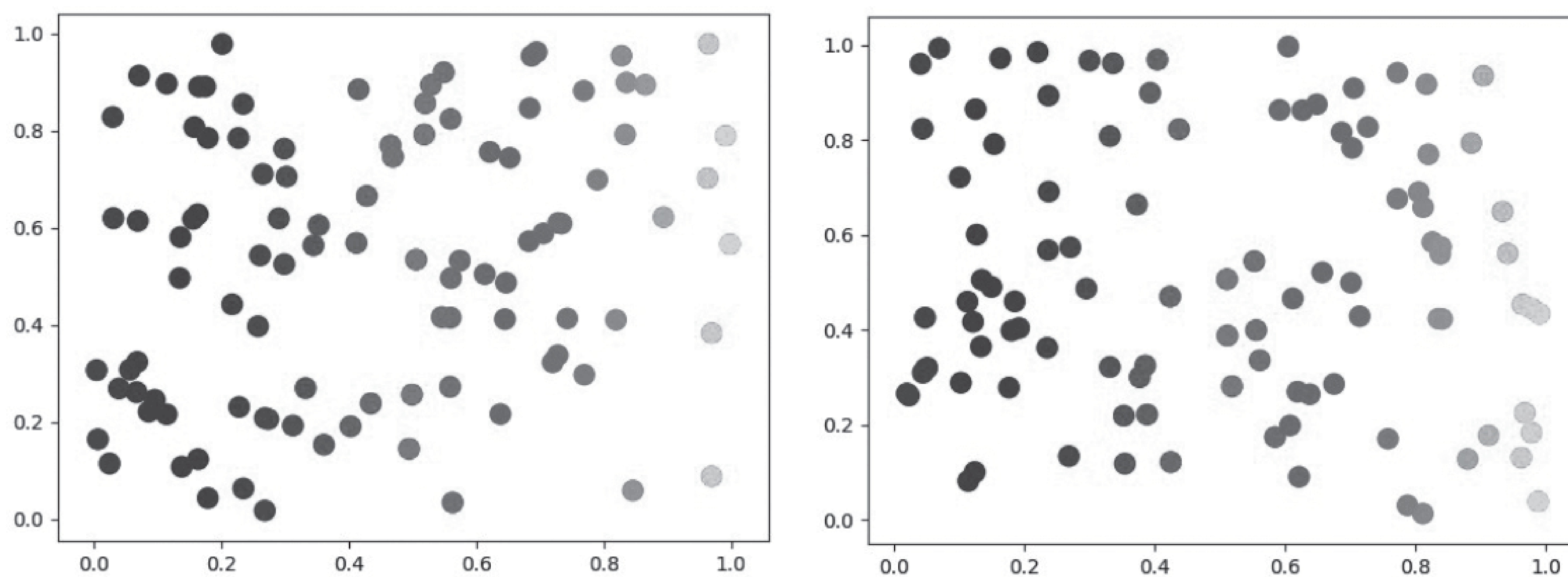
```
np.random.random(N)          # 返回 N 个 0.0 ~ 1.0 的数字
```

20-4-1 一个简单的应用

程序实例 ch20_21.py：产生 100 个 0.0 ～ 1.0 的随机数，第 10 行的 cmp='brg' 意义是使用 brg 色彩映射表绘制这个图表，色彩会随 x 轴变化。当关闭图表时，会询问是否继续，如果输入 n/N 则结束。其实因为数据是随机数，所以每次都可产生不同的效果。

```
1  # ch20_21.py
2  import matplotlib.pyplot as plt
3  import numpy as np
4
5  num = 100
6  while True:
7      x = np.random.random(100)                # 可以产生num个0.0~1.0的数字
8      y = np.random.random(100)
9      t = x                                    # 色彩随x轴变化
10     plt.scatter(x, y, s=100, c=t, cmap='brg')
11     plt.show()
12     yORn = input("是否继续 ?(y/n) ")          # 询问是否继续
13     if yORn == 'n' or yORn == 'N':           # 输入n或N则程序结束
14         break
```

执行结果



上述程序笔者使用第 5 行的 num 控制产生随机数的数量，其实读者可以自行修订，增加或减少随机数的数量，以体会本程序的运作。

20-4-2 随机数的移动

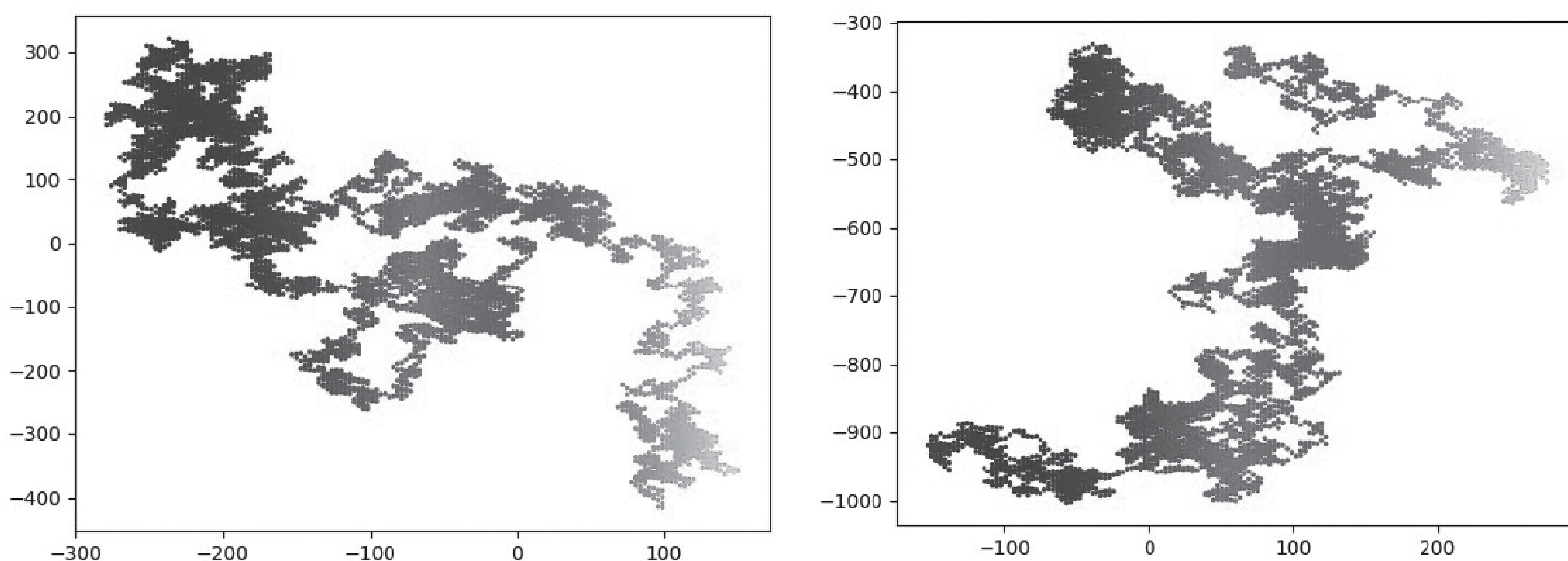
其实也可以针对随机数的特性，让每个点随着随机数的变化产生随机移动，经过大量的运算后，每次均可产生不同但有趣的图形。

程序实例 ch20_22.py：随机数移动的程序设计，这个程序在设计时，最初点的起始位置是 (0,0)，

程序第 7 行可以设置下一个点的 x 轴是往右移动 3 或是往左移动 3，程序第 9 行可以设置下一个点的 y 轴是往上移动 1 或 5 或是往下移动 1 或 5。每此执行完 10000 个点的测试后，会询问是否继续。如果继续，先将上一回合的终点坐标当作新回合的起点坐标（27、28 行），然后清除列表索引 x[0] 和 y[0] 以外的元素（29、30 行）。

```
1  # ch20_22.py
2  import matplotlib.pyplot as plt
3  import random
4
5  def loc(index):
6      ''' 处理坐标的移动 '''
7      x_mov = random.choice([-3, 3])              # 随机x轴移动值
8      xloc = x[index-1] + x_mov                   # 计算x轴新位置
9      y_mov = random.choice([-5, -1, 1, 5])       # 随机y轴移动值
10     yloc = y[index-1] + y_mov                   # 计算y轴新位置
11     x.append(xloc)                              # x轴新位置加入列表
12     y.append(yloc)                              # y轴新位置加入列表
13
14 num = 10000                                     # 设置随机点的数量
15 x = [0]                                         # 设置第一次执行x坐标
16 y = [0]                                         # 设置第一次执行y坐标
17 while True:
18     for i in range(1, num):                     # 建立点的坐标
19         loc(i)
20     t = x                                       # 色彩随x轴变化
21     plt.scatter(x, y, s=2, c=t, cmap='brg')
22     plt.show()
23     yORn = input("是否继续 ?(y/n) ")            # 询问是否继续
24     if yORn == 'n' or yORn == 'N':              # 输入n或N则程序结束
25         break
26     else:
27         x[0] = x[num-1]                         # 上次结束x坐标成新的起点x坐标
28         y[0] = y[num-1]                         # 上次结束y坐标成新的起点y坐标
29         del x[1:]                               # 删除旧列表x坐标元素
30         del y[1:]                               # 删除旧列表y坐标元素
```

执行结果



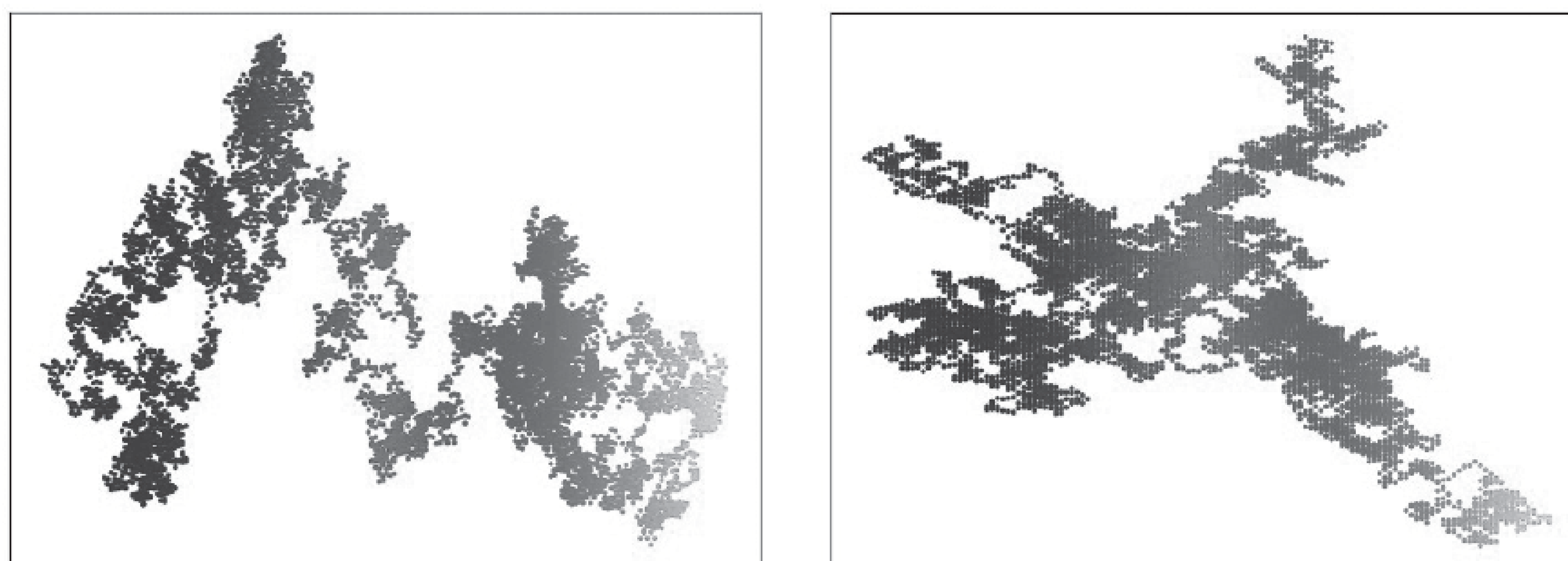
20-4-3 隐藏坐标

有时候我们设计随机数移动建立了漂亮的图案后，觉得坐标好像很煞风景，可以使用下列程序实例 ch20_23.py 内的 axes( ).get_xaxis( )、axes( ).get_yaxis( )、set_visible( ) 方法隐藏坐标。

程序实例 ch20_23.py：重新设计 ch20_22.py 隐藏坐标，这个程序只是增加下列行。

```
22     plt.axes().get_xaxis().set_visible(False)    # 隐藏x轴坐标
23     plt.axes().get_yaxis().set_visible(False)    # 隐藏y轴坐标
```

执行结果



20-5 绘制多个图表

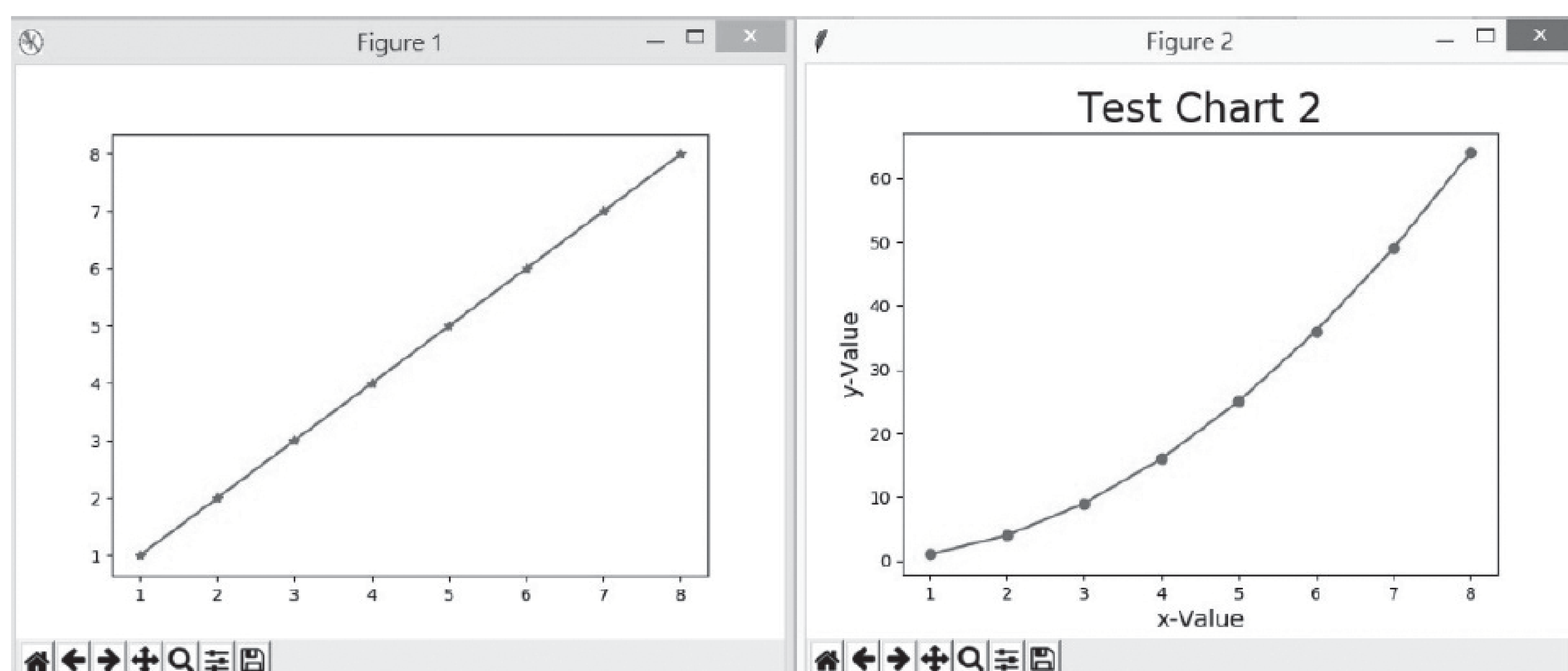
20-5-1 一个程序有多个图表

Python 允许一个程序绘制多个图表，默认是一个程序绘制一个图表（Figure），如果想要绘制多个图表，可以使用 figure（N）设置图表，N 是图表的序号。在建立多个图表时，只要将所要绘制的图接在欲放置的图表后面即可。

程序实例 ch20_24.py：设计两个图表，将 data1 线条放在图表 Figure 1，将 data2 线条放在图表 Figure 2。同时图表 Figure 2 将会建立图表标题与 x 轴和 y 轴的标题。

```
1  # ch20_24.py
2  import matplotlib.pyplot as plt
3
4  data1 = [1, 2, 3, 4, 5, 6, 7, 8]                # data1线条
5  data2 = [1, 4, 9, 16, 25, 36, 49, 64]           # data2线条
6  seq = [1, 2, 3, 4, 5, 6, 7, 8]
7  plt.figure(1)                                   # 建立图表1
8  plt.plot(seq, data1, '-*')                      # 绘制图表1
9  plt.figure(2)                                   # 建立图表2
10 plt.plot(seq, data2, '-o')                      # 以下皆是绘制图表2
11 plt.title("Test Chart 2", fontsize=24)
12 plt.xlabel("x-Value", fontsize=14)
13 plt.ylabel("y-Value", fontsize=14)
14 plt.show()
```

执行结果



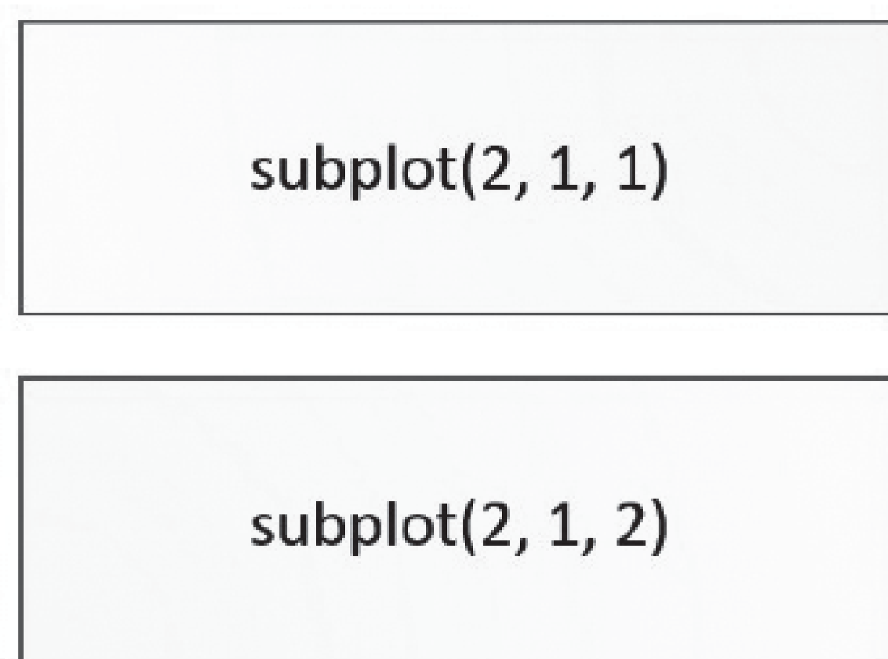
上述第 8 行所绘制的 data1 图表因为是接在 plt.figure(1) 后面，所以所绘制的图出现在 Figure 1。上述第 10 ～ 13 行所绘制的 data2 图表因为是接在 plt.figure(2) 后面，所以所绘制的图出现在 Figure 2。

20-5-2　含有子图的图表

要设计含有子图的图表需要使用 subplot( ) 方法，语法如下。

```
subplot(x1, x2, x3)
```

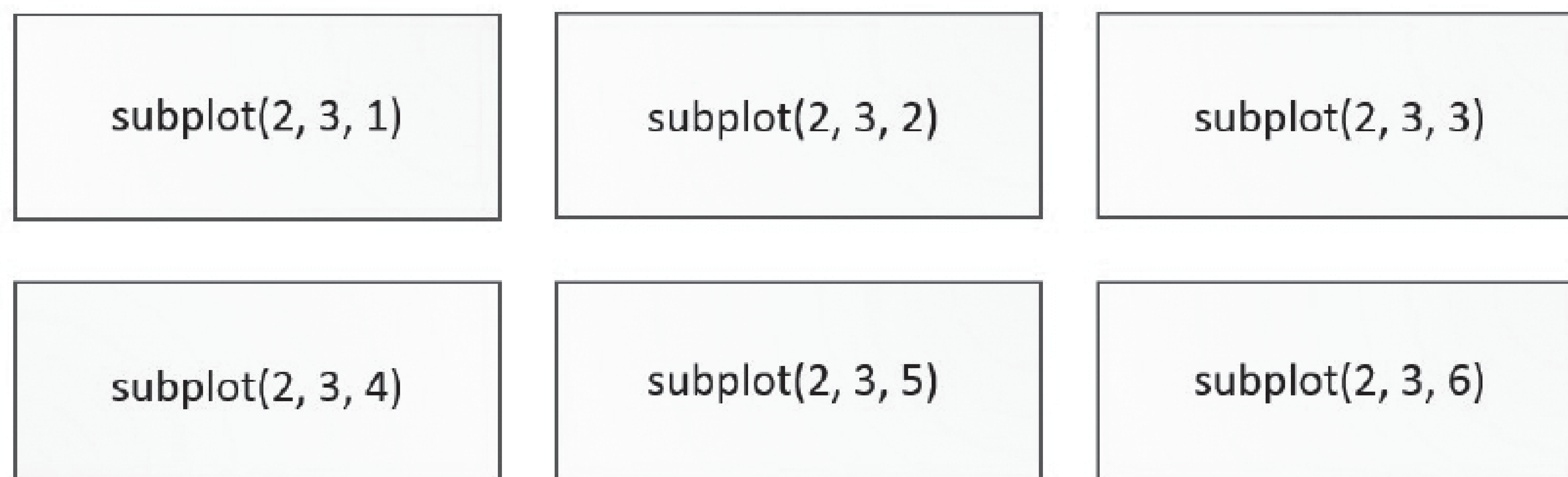
x1 代表上下（垂直）要绘制几张图，x2 代表左右（水平）要绘制几张图。x3 代表这是第几张图。如果规划是一个 Figure 绘制上下两张图，那么 subplot( ) 的应用如下。



如果规划是一个 Figure 绘制左右两张图，那么 subplot( ) 的应用如下。



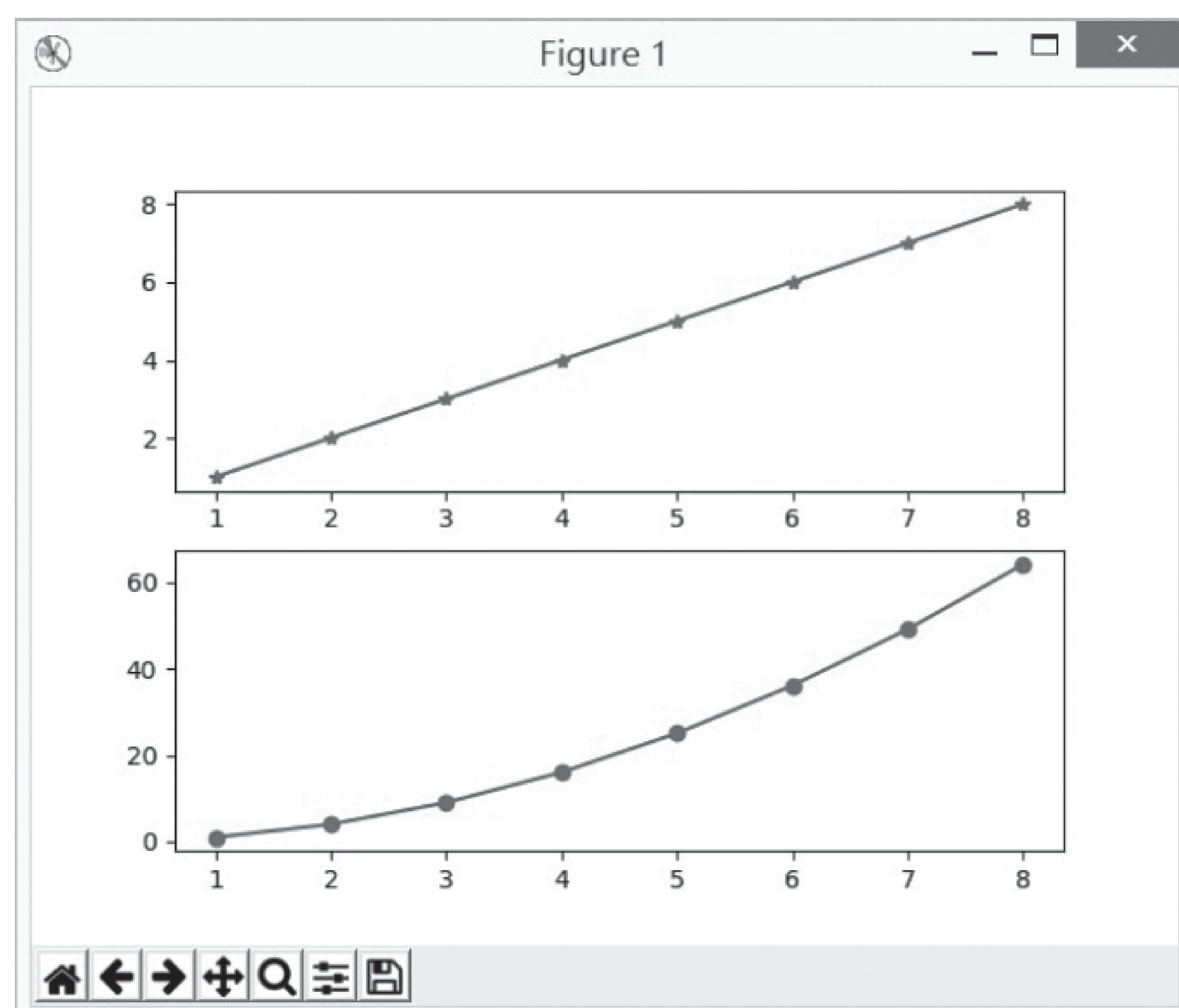
如果规划是一个 Figure 绘制上下两张图，左右三张图，那么 subplot( ) 的应用如下。



程序实例 ch20_25.py：在一个 Figure 内绘制上下子图的应用。

```
 1  # ch20_25.py
 2  import matplotlib.pyplot as plt
 3
 4  data1 = [1, 2, 3, 4, 5, 6, 7, 8]                # data1线条
 5  data2 = [1, 4, 9, 16, 25, 36, 49, 64]           # data2线条
 6  seq = [1, 2, 3, 4, 5, 6, 7, 8]
 7  plt.subplot(2, 1, 1)                            # 子图1
 8  plt.plot(seq, data1, '-*')
 9  plt.subplot(2, 1, 2)                            # 子图2
10  plt.plot(seq, data2, '-o')
11  plt.show()
```

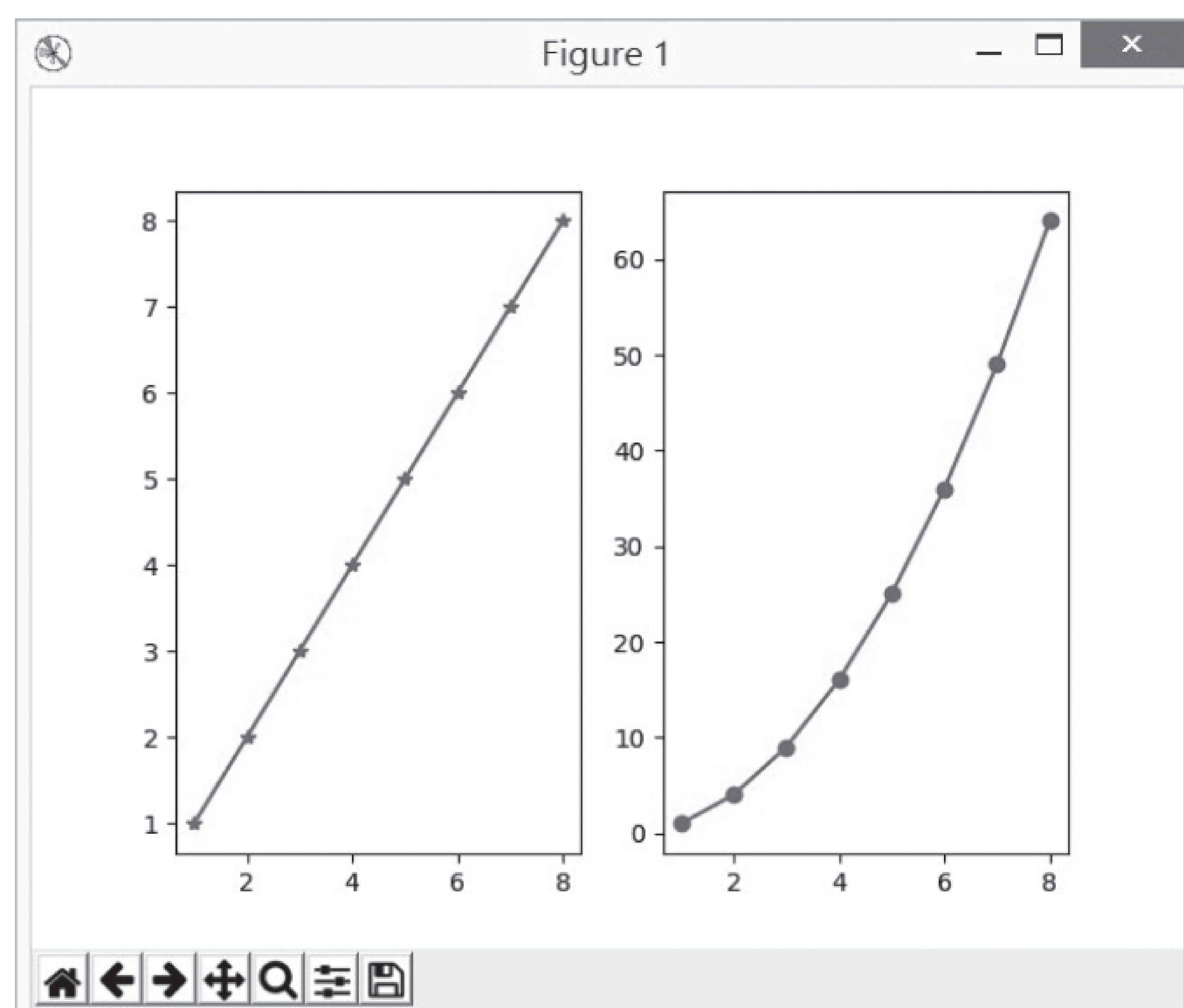
执行结果



程序实例 ch20_26.py：在一个 Figure 内绘制左右子图的应用。

```
 1  # ch20_26.py
 2  import matplotlib.pyplot as plt
 3
 4  data1 = [1, 2, 3, 4, 5, 6, 7, 8]                 # data1线条
 5  data2 = [1, 4, 9, 16, 25, 36, 49, 64]            # data2线条
 6  seq = [1, 2, 3, 4, 5, 6, 7, 8]
 7  plt.subplot(1, 2, 1)                             # 子图1
 8  plt.plot(seq, data1, '-*')
 9  plt.subplot(1, 2, 2)                             # 子图2
10  plt.plot(seq, data2, '-o')
11  plt.show()
```

执行结果



20-6 直方图的制作

20-6-1 bar( )

在直方图的制作中，可以使用 bar( ) 方法，常用的语法如下。

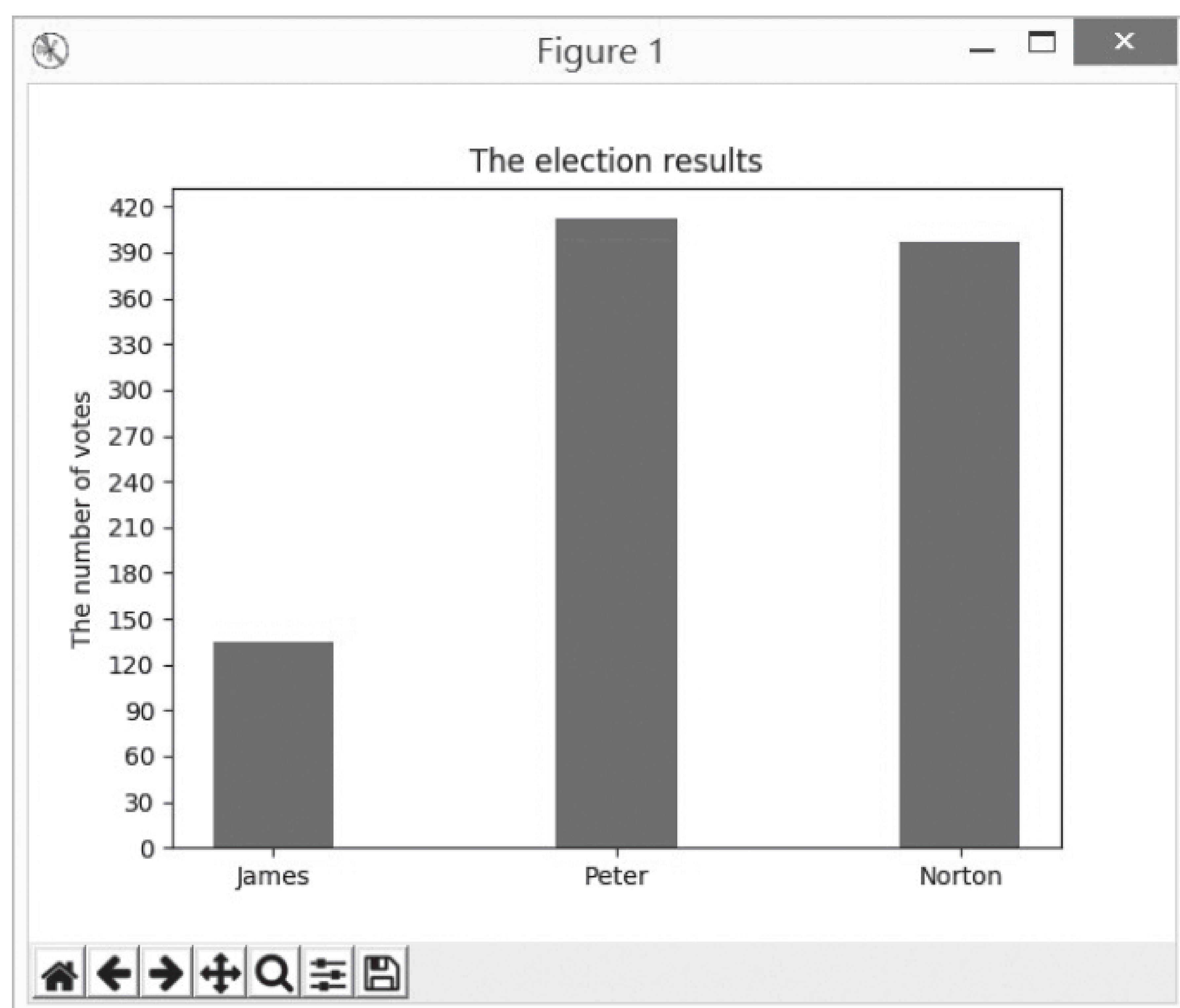
bar(x, y, width)

x 是一个**列表**，主要是直方图 x 轴位置，y 也是**列表**，代表 y 轴的值，width 是直方图的宽度，默认是 0.85。至于其他绘图参数也可以在此使用，例如，xlabel（x 轴标题）、ylabel（y 轴标题）、xticks（x 轴刻度）、yticks（y 轴刻度）、color（颜色）、lengend（图例）。

程序实例 ch20_27.py：有一个选举，James 得票 135、Peter 得票 412、Norton 得票 397，用直方图表示。

```
1  # ch20_27.py
2  import numpy as np
3  import matplotlib.pyplot as plt
4
5  votes = [135, 412, 397]          # 得票数
6  N = len(votes)                   # 计算长度
7  x = np.arange(N)                 # 直方图x轴坐标
8  width = 0.35                     # 直方图宽度
9  plt.bar(x, votes, width)         # 绘制直方图
10
11 plt.ylabel('The number of votes')
12 plt.title('The election results')
13 plt.xticks(x, ('James', 'Peter', 'Norton'))
14 plt.yticks(np.arange(0, 450, 30))
15 plt.show()
```

执行结果

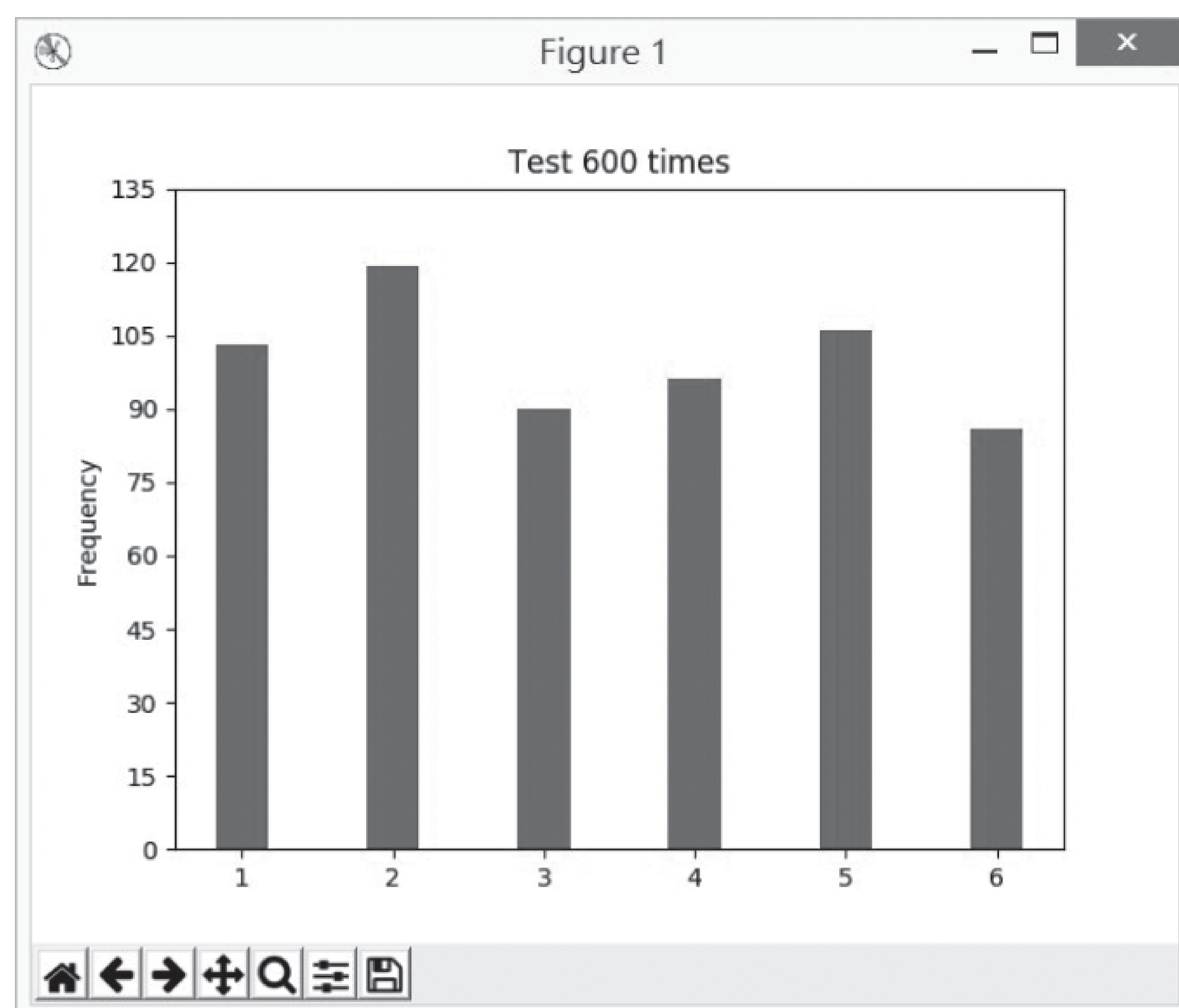


上述程序第 11 行是设置 y 轴的标题，第 12 行是设置直方图的标题，第 13 行则是设置 x 轴刻度，第 14 行是设置 y 轴刻度。

程序实例 ch20_28.py：掷骰子的概率设计。一个骰子有 6 面，分别记载 1、2、3、4、5、6，这个程序会用随机数计算 600 次每个数字出现的次数，同时用柱形图表示，为了让读者有不同体验，笔者将图表颜色改为绿色。

```
1   # ch20_28.py
2   import numpy as np
3   import matplotlib.pyplot as plt
4   from random import randint
5
6   def dice_generator(times, sides):
7       ''' 处理随机数 '''
8       for i in range(times):
9           ranNum = randint(1, sides)          # 产生1~6随机数
10          dice.append(ranNum)
11  def dice_count(sides):
12      '''计算1~6出现次数'''
13      for i in range(1, sides+1):
14          frequency = dice.count(i)           # 计算i出现在dice列表的次数
15          frequencies.append(frequency)
16
17  times = 600                                 # 掷骰子次数
18  sides = 6                                   # 骰子有几面
19  dice = []                                   # 建立掷骰子的列表
20  frequencies = []                            # 存储每一面骰子出现次数列表
21  dice_generator(times, sides)                # 产生掷骰子的列表
22  dice_count(sides)                           # 将骰子列表转成次数列表
23  x = np.arange(6)                            # 直方图x轴坐标
24  width = 0.35                                # 直方图宽度
25  plt.bar(x, frequencies, width, color='g')   # 绘制直方图
26  plt.ylabel('Frequency')
27  plt.title('Test 600 times')
28  plt.xticks(x, ('1', '2', '3', '4', '5', '6'))
29  plt.yticks(np.arange(0, 150, 15))
30  plt.show()
```

执行结果



上述程序最重要的是第 11 ～ 15 行的 dice_count( ) 函数，这个函数主要是将含 600 个元素的 dice 列表，分别计算 1、2、3、4、5、6 各出现的次数，然后将结果存储至 frequencies 列表。如果读者忘记 count( ) 的用法，可以参考 6-6-2 节。

20-6-2 hist( )

这也是一个直方图的制作方法，特别适合为统计分布数据绘图，语法如下。

```
h = hist(x, bins, color, options …)          # 返回值 h 可有可无
```

在此只介绍常用的参数，x 是一个列表或数组（23 章会介绍数组），是每个 bins 分布的数据。bins 则是箱子（可以想成长条）的个数或是可想成组别个数。color 则是设置长条颜色。options 有许多，density 可以是 True 或 False，如果是 True 表示 y 轴呈现的是占比，每个直方条状的占比总和是 1。

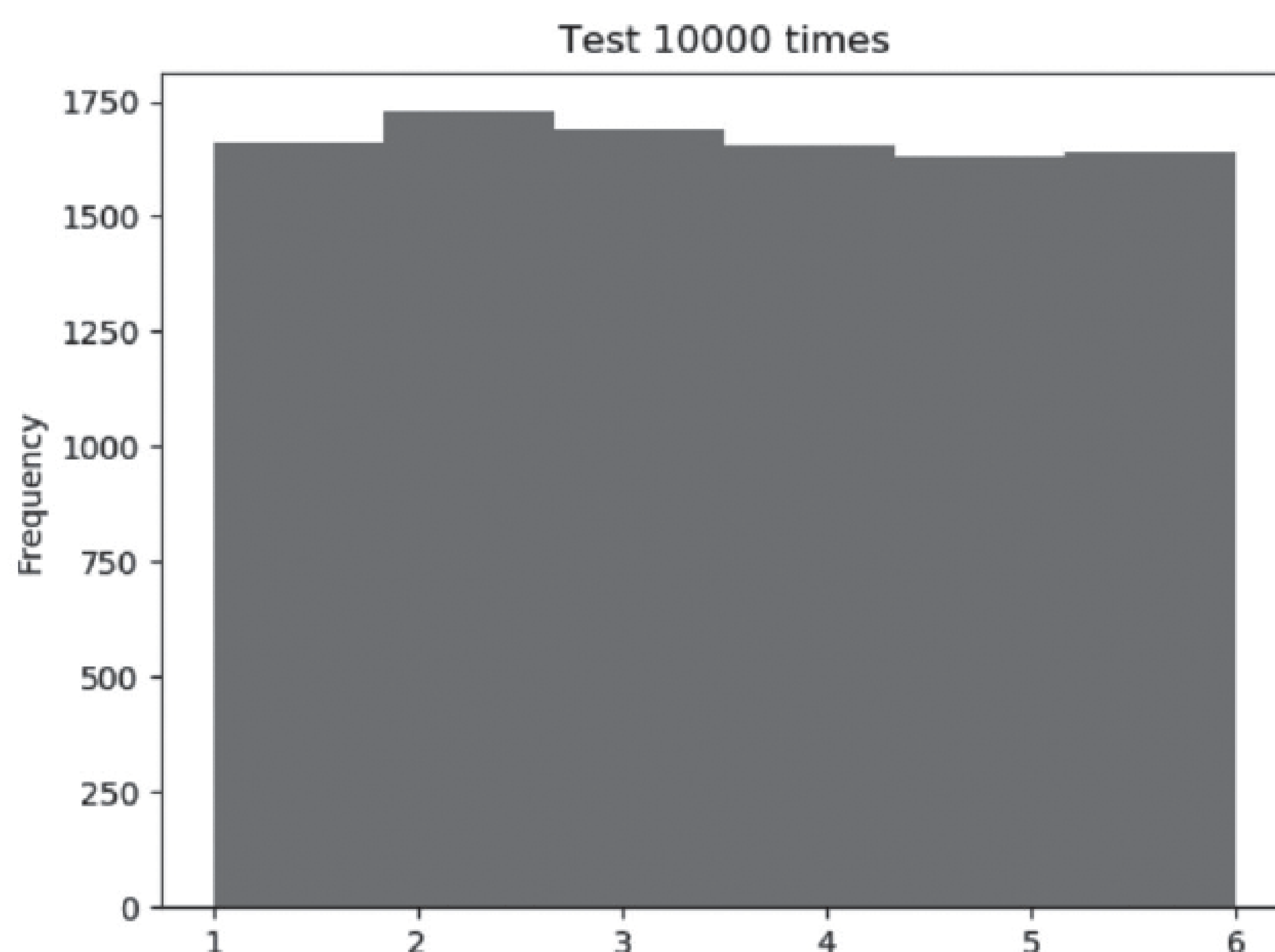
返回值 h 是元组，可以不理会，如果有设置返回值，则 h 值所返回的 h[0] 是 bins 的数量数组，每个索引记载这个 bins 的 y 轴值，由索引数量也可以知道 bins 的数量，相当于是直方长条数。h[1] 也是数组，此数组记载 bins 的 x 轴值。第 23 章将说明更多数组的知识。

程序实例 ch20_28_1.py：以 hist 直方图打印掷骰子 10000 次的结果，需留意由于是随机数产生骰子的 6 个面，所以每次执行结果皆会不相同，这个程序同时列出 hist( ) 的返回值，也就是骰子出现的次数。

```
1  # ch20_28_1.py
2  import numpy as np
3  import matplotlib.pyplot as plt
4  from random import randint
5
6  def dice_generator(times, sides):
7      ''' 处理随机数 '''
8      for i in range(times):
9          ranNum = randint(1, sides)                  # 产生1~6随机数
10         dice.append(ranNum)
11
12 times = 10000                                       # 掷骰子次数
13 sides = 6                                           # 骰子有几面
14 dice = []                                           # 建立掷骰子的列表
15 dice_generator(times, sides)                        # 产生掷骰子的列表
16
17 h = plt.hist(dice, sides)                           # 绘制hist图
18 print("bins的y轴 ",h[0])
19 print("bins的x轴 ",h[1])
20 plt.ylabel('Frequency')
21 plt.title('Test 10000 times')
22 plt.show()
```

执行结果

```
==================== RESTART: D:\Python\ch20\ch20_28_1.py ====================
bins的y轴  [1661. 1637. 1615. 1713. 1684. 1690.]
bins的x轴  [1.         1.83333333 2.66666667 3.5        4.33333333 5.16666667
 6.        ]
```



20-7 圆饼图的制作 pie( )

在圆饼图的制作中，可以使用 pie( ) 方法，语法如下。

pie(x, options, …)

x 是一个列表，主要是圆饼图 x 轴的数据，options 代表系列选择性参数，可以是下列参数内容。

labels：圆饼图项目所组成的列表。

colors：圆饼图项目颜色所组成的列表，如果省略则用默认颜色。

explode：可设置是否从圆饼图分离的列表，0 表示不分离，一般可用 0.1 分离，数值越大分离越远。例如，读者在程序实例 ch20_29.py 中可改用 0.2 测试，效果不同，默认是 0。

autopct：表示项目的百分比格式，基本语法是 % 格式 %%。例如，%2.2%% 表示整数 2 位数，小数两位数。

labeldistance：项目标题与圆饼图中心的距离是半径的多少倍。例如，1.2 代表是 1.2 倍。

center：圆中心坐标，默认是 0。

shadow：True 表示圆饼图形有阴影，False 表示圆饼图形没有阴影，默认是 False。

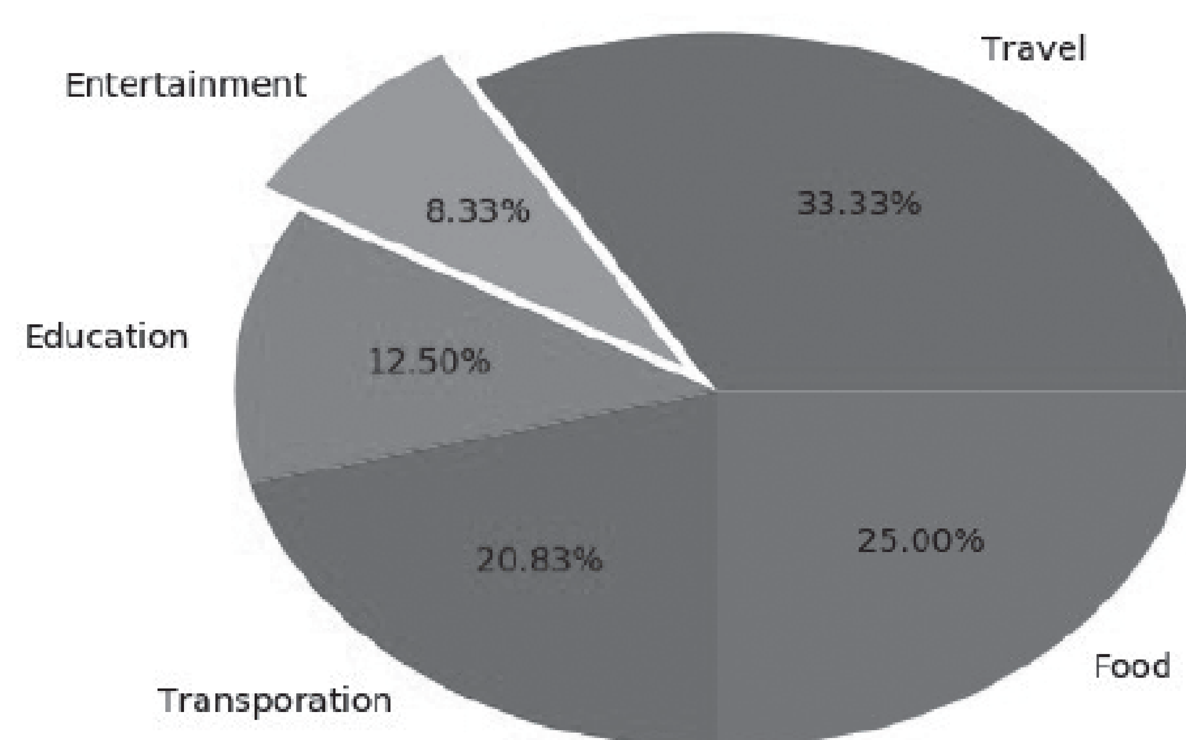
程序实例 ch20_29.py：有一个家庭开支的费用如下，然后设计此圆饼图。

旅行 (Travel):8000　　娱乐 (Entertainment):2000

教育 (Education):3000　　交通 (Transporation):5000　　餐费 (Food):6000

```
1  # ch20_29.py
2  import matplotlib.pyplot as plt
3
4  sorts = ["Travel","Entertainment","Education","Transporation","Food"]
5  fee = [8000,2000,3000,5000,6000]
6
7  plt.pie(fee,labels=sorts,explode=(0,0.3,0,0,0),
8          autopct="%1.2f%%")      # 绘制圆饼图
9  plt.show()
```

执行结果



上述程序第 7 行的 explode=（0,0.1,0,0,0）相当于是第 2 个数据做分离效果。

20-8 图表显示中文

一个图表无法显示中文，坦白地说读者内心一定感觉有缺憾，至少笔者感觉如此。matplotlib 无法显示中文主要原因在于安装此模块时所配置的文件。

```
~Python36\Lib\site-packages\matplotlib\mpl-data\matplotlibrc
```

在此文件内的 font_sans-serif 没有配置中文字体，我们可以在此字段增加中文字体，但是笔者不鼓励更改系统内建文件。笔者将使用动态配置方式处理，让图表显示中文字体。其实可以在程序内增加下列程序代码，用 rcParams( ) 方法为 matplotply 配置中文字体参数，就可以显示中文了。

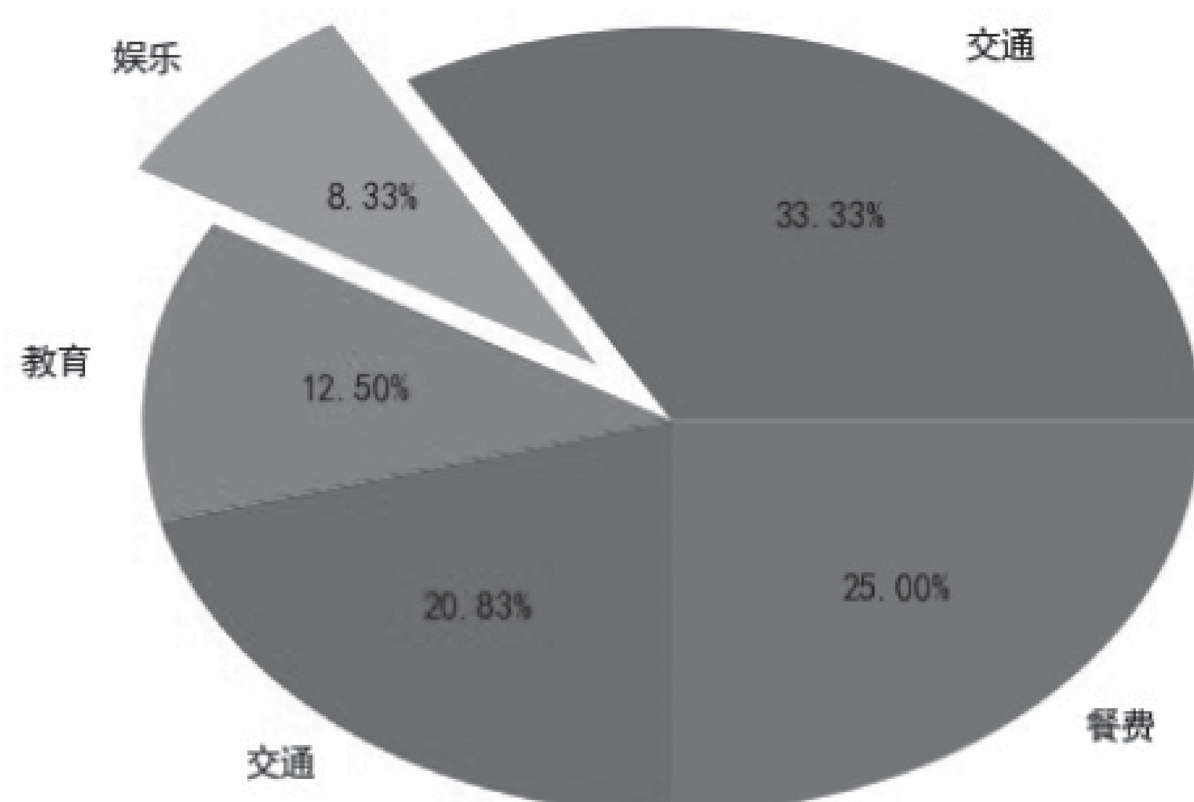
```
from pylab import mlp                            # matplotlib 的子模块
mlp.rcParams["font.sans-serif"] = ["SimHei"]     # 黑体
mlp.rcParams["axes.unicode_minus"] = False       # 让其可以显示负号
```

另外，每个要显示的中文字符串需要在字符串前加上 u" **中文字符串** "。

程序实例 ch20_30.py：重新设计 ch20_29.py，以中文显示各项花费。

```
1  # ch20_30.py
2  import matplotlib.pyplot as plt
3  from pylab import mpl
4
5  mpl.rcParams["font.sans-serif"] = ["SimHei"]        # 使用黑体
6  mpl.rcParams["axes.unicode_minus"] = False          # 让其可以显示负号
7
8  sorts = [u"交通",u"娱乐",u"教育",u"交通",u"餐费"]
9  fee = [8000,2000,3000,5000,6000]
10
11 plt.pie(fee,labels=sorts,explode=(0,0.2,0,0,0),
12         autopct="%1.2f%%")                          # 绘制圆饼图
13 plt.show()
```

执行结果



20-9 专题——股市数据读取与图表制作

这一节将介绍使用 twstock 模块读取中国台湾股票信息，同时利用本节知识建立折线图表。使用前需安装 twstock 模块。

```
pip install twstock
```

读者可以参考下列网址，了解更完整的信息。

https://twstock.readthedocs.io/zh_TW/latest/

20-9-1 Stock( ) 建构元

可以使用 Stock( ) 建构元传入股票代号，然后可以返回此股票代号的 Stock 对象。中国台湾著名的股票台积电的股票代号是 2330，如果输入 2330，即可以获得台积电的股票的对象。

```
>>> import twstock
>>> stock2330 = twstock.Stock("2330")
```

20-9-2　Stock 对象属性

有了前一节的 Stock 对象后，可以参考下表获得对象属性。

Stock 对象属性	说明
sid	股票代号字符串
open	近 31 天的开盘价（元）列表
high	近 31 天的最高价（元）列表
low	近 31 天的最低价（元）列表
close 或 price	近 31 天的收盘价（元）列表
capacity	近 31 天的成交量（股）列表
transaction	近 31 天的成交笔数（笔）列表
turnover	近 31 天的成交金额（元）列表
change	近 31 天的涨跌幅（元）列表
date	近 31 天的交易日期 datetime 对象列表
data	近 31 天的 Stock 对象全部数据内容列表
raw_data	近 31 天的原始数据列表

程序实例 ch20_31.py：获得台积电股票代号和近 31 天的收盘价。

```
1  # ch20_31.py
2  import twstock
3  stock2330 = twstock.Stock("2330")
4
5  print("股票代号    : ", stock2330.sid)
6  print("股票收盘价 : ", stock2330.price)
```

执行结果

```
===================== RESTART: D:\Python\ch20\ch20_31.py =====================
股票代号    :  2330
股票收盘价 :  [266.0, 268.0, 269.0, 267.5, 260.0, 259.5, 259.0, 259.0, 265.0, 25
9.0, 262.5, 260.0, 256.5, 256.0, 250.5, 248.5, 249.0, 247.0, 241.5, 238.0, 234.0
, 238.0, 230.0, 233.0, 231.0, 230.5, 229.5, 231.0, 235.5, 238.0, 233.0]
```

在所返回的 31 天收盘价列表中，[0] 是最早的收盘价，[30] 是前一个交易日的收盘价。

实例 1：返回台积电 31 天前的收盘价与前一个交易日的收盘价。

```
>>> import twstock
>>> stock2330 = twstock.Stock("2330")
>>> stock2330.price[0]
222.5
>>> stock2330.price[30]
221.0
```

实例 2：了解台积电 data 属性的全部内容，可以使用 data 属性，此例只是列出部分内容。

```
>>> import twstock
>>> stock2330 = twstock.Stock("2330")
>>> stock2330.data
[Data(date=datetime.datetime(2018, 12, 18, 0, 0), capacity=30270541, turnover=67
13448829, open=221.0, high=223.0, low=220.5, close=222.5, change=-1.0, transacti
on=10521), Data(date=datetime.datetime(2018, 12, 19, 0, 0), capacity=23415082, t
```

在 Stock 属性中除了股票代号 sid 是返回字符串外，其他都是返回列表，此时可以使用切片方式处理。

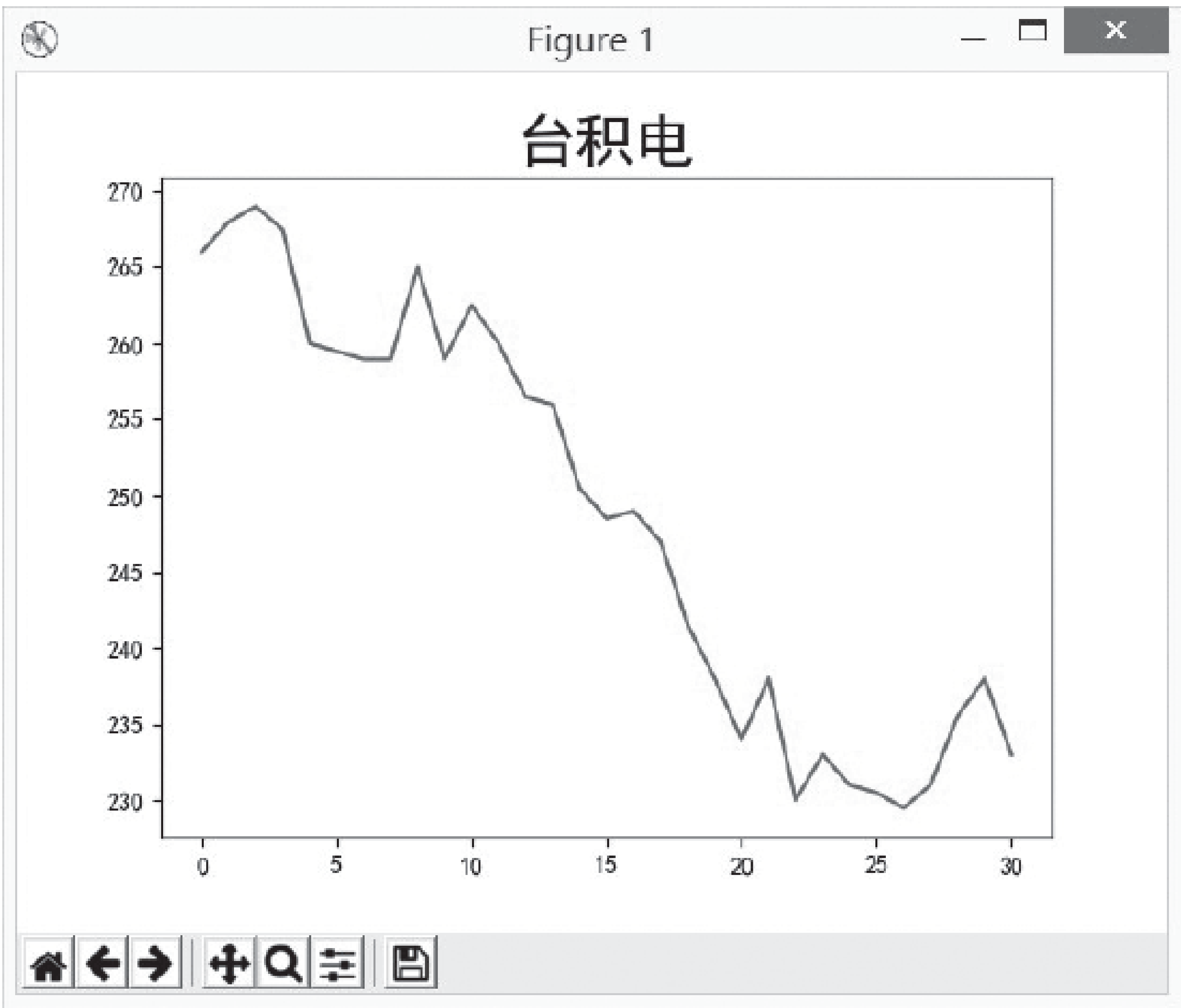
实例 3：返回台积电近 5 天的股票收盘价。

```
>>> import twstock
>>> stock2330 = twstock.Stock("2330")
>>> stock2330.price[-5:]
[222.5, 226.0, 229.0, 222.5, 221.0]
```

程序实例 ch20_32.py：列出近 31 天台积电收盘价的折线图。

```
1  # ch20_32.py
2  import matplotlib.pyplot as plt
3  from pylab import mpl
4  import twstock
5
6  mpl.rcParams["font.sans-serif"] = ["SimHei"]        # 使用黑体
7
8  stock2330 = twstock.Stock("2330")
9  plt.title(u"台积电", fontsize=24)
10 plt.plot(stock2330.price)
11 plt.show()
```

执行结果



20-9-3 Stock 对象方法

有了 20-9-1 节的 Stock 对象后，可以参考下表获得对象方法。

Stock 对象方法	说明
fetch_31( )	最近 31 天的事务数据（Data 对象）列表
fetch(year, month)	指定年月的事务数据（Data 对象）列表
fetch_from(year, month)	指定年月至今的事务数据（Data 对象）列表
moving_average(data,days)	列表数据 data 的 days 日的平均值列表
continuous(data)	列表 data 持续上涨天数

实例 1：返回 2018 年 1 月的台积电股市事务数据，下面只列出部分结果。

```
>>> import twstock
>>> stock2330 = twstock.Stock("2330")
>>> stock2330.fetch(2018, 1)
[Data(date=datetime.datetime(2018, 1, 2, 0, 0), capacity=18055269, turnover=4188
555408, open=231.5, high=232.5, low=231.0, close=232.5, change=3.0, transaction=
9954), Data(date=datetime.datetime(2018, 1, 3, 0, 0), capacity=31706091, turnove
```

实例 2：延续上一个实例，返回 2018 年 1 月台积电收盘价格数据。

```
>>> stock2330.price
[232.5, 237.0, 239.5, 240.0, 242.0, 242.0, 236.5, 235.0, 237.0, 240.0, 240.5, 24
2.0, 248.5, 255.5, 261.5, 266.0, 258.0, 258.0, 255.0, 258.5, 253.0, 255.0]
```

方法 moving_average(data,days) 是返回均线列表，这个方法需要两个参数，days 天数是代表几天均线，例如，若第 2 个参数是 5，则代表 5 天均线值。所谓的 5 天均线值是指第 0 ～ 4 个数据的平均值当作第 0 个，第 1 ～ 5 个数据的平均值当作第 1 个，所以均线数据列表元素会比较少。

实例 3：延续上一个实例，返回 2018 年 1 月台积电收盘价格数据的 5 天均线数据列表。

```
>>> ave5 = stock2330.moving_average(stock2330.price, 5)
>>> ave5
[238.2, 240.1, 240.0, 239.1, 238.5, 238.1, 237.8, 238.9, 241.6, 245.3, 249.6, 25
4.7, 257.9, 259.8, 259.7, 259.1, 256.5, 255.9]
```

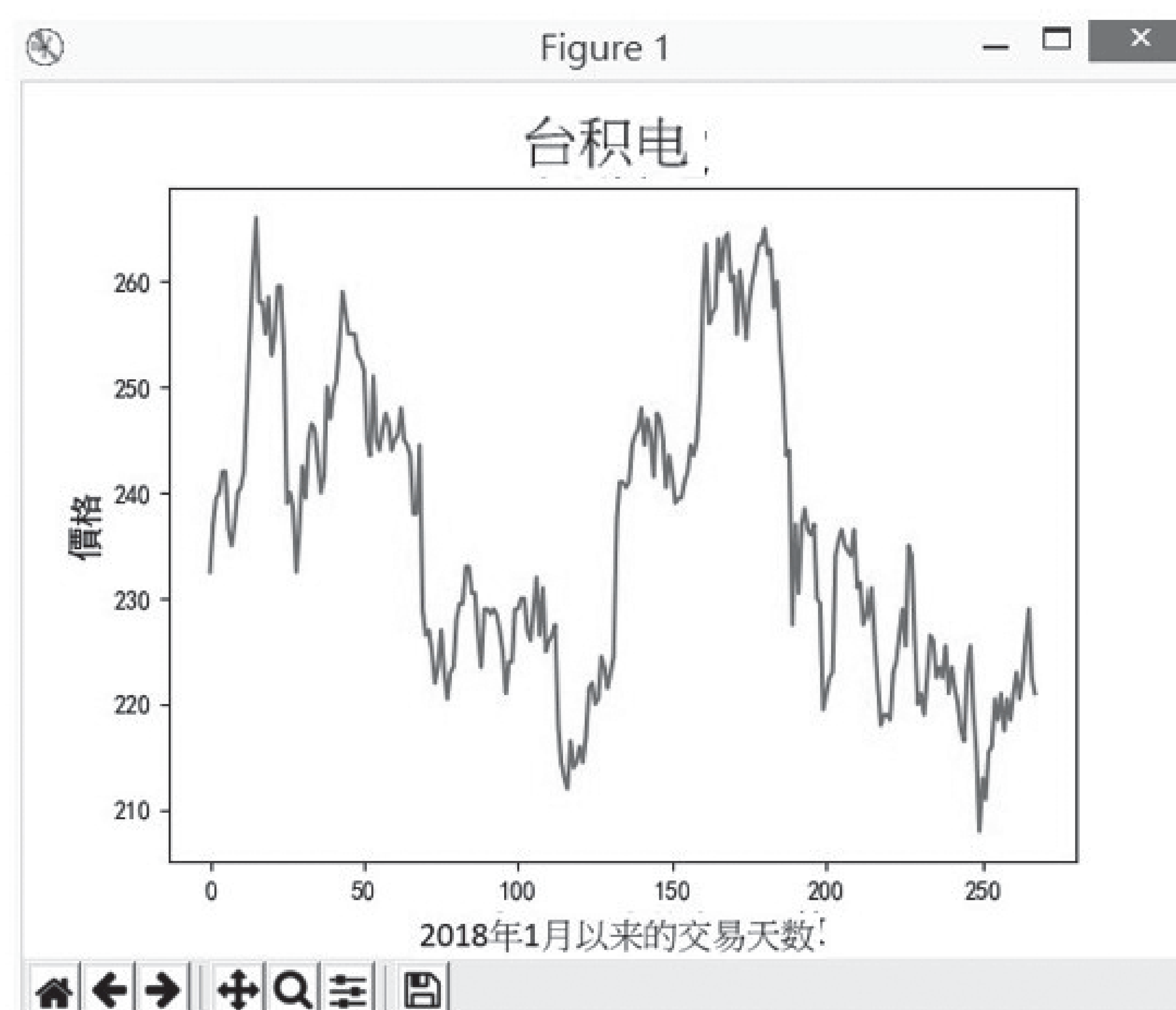
实例 4：延续上一个实例，返回 2018 年 1 月台积电收盘价格数据的 5 天均线数据列表的连续上涨天数。留意：每天做比较，如果上涨会加 1，下跌会减 1。

```
>>> stock2330.continuous(ave5)
-4
```

程序实例 ch20_33.py：以折线图打印台积电 2018 年 1 月以来的收盘价格数据。

```
1  # ch20_33.py
2  import matplotlib.pyplot as plt
3  from pylab import mpl
4  import twstock
5
6  mpl.rcParams["font.sans-serif"] = ["SimHei"]        # 使用黑体
7
8  stock2330 = twstock.Stock("2330")
9  stock2330.fetch_from(2018,1)
10 plt.title(u"台积电", fontsize=24)
11 plt.xlabel(u"2018年1月以来的交易天数", fontsize=14)
12 plt.ylabel(u"价格", fontsize=14)
13 plt.plot(stock2330.price)
14 plt.show()
```

执行结果



20-9-4 取得单一股票的实时数据

在使用 twstock 模块时，可以使用 realtime.get( ) 取得特定股票的实时信息，这些信息包含股票代号 code、名称 name、全名 fullname、收盘时间 time 等。同时有包含目前 5 个买进和卖出的金额与数量。

实例 1：列出台积电的实时数据。

```
>>> import twstock
>>> stock2330 = twstock.realtime.get('2330')
>>> stock2330
{'timestamp': 1548829800.0, 'info': {'code': '2330', 'channel': '2330.tw', 'name
': '台積電', 'fullname': '台灣積體電路製造股份有限公司', 'time': '2019-01-30 14:
30:00'}, 'realtime': {'latest_trade_price': '221.00', 'trade_volume': '8683', 'a
ccumulate_trade_volume': '44721', 'best_bid_price': ['220.50', '220.00', '219.50
', '219.00', '218.50'], 'best_bid_volume': ['847', '3366', '1408', '1964', '1602
'], 'best_ask_price': ['221.00', '221.50', '222.00', '222.50', '223.00'], 'best_
ask_volume': ['308', '4094', '1666', '474', '483'], 'open': '220.50', 'high': '2
21.50', 'low': '220.00'}, 'success': True}
```

实例 2：延续前一实例，列出目前 5 个买进金额与数量。

```
>>> stock2330["realtime"]["best_bid_price"]    #5个买进金额
['220.50', '220.00', '219.50', '219.00', '218.50']
>>> stock2330["realtime"]["best_bid_volume"]   #5个买进数量
['847', '3366', '1408', '1964', '1602']
```

实例 3：延续前一实例，列出目前 5 个卖出金额与数量。

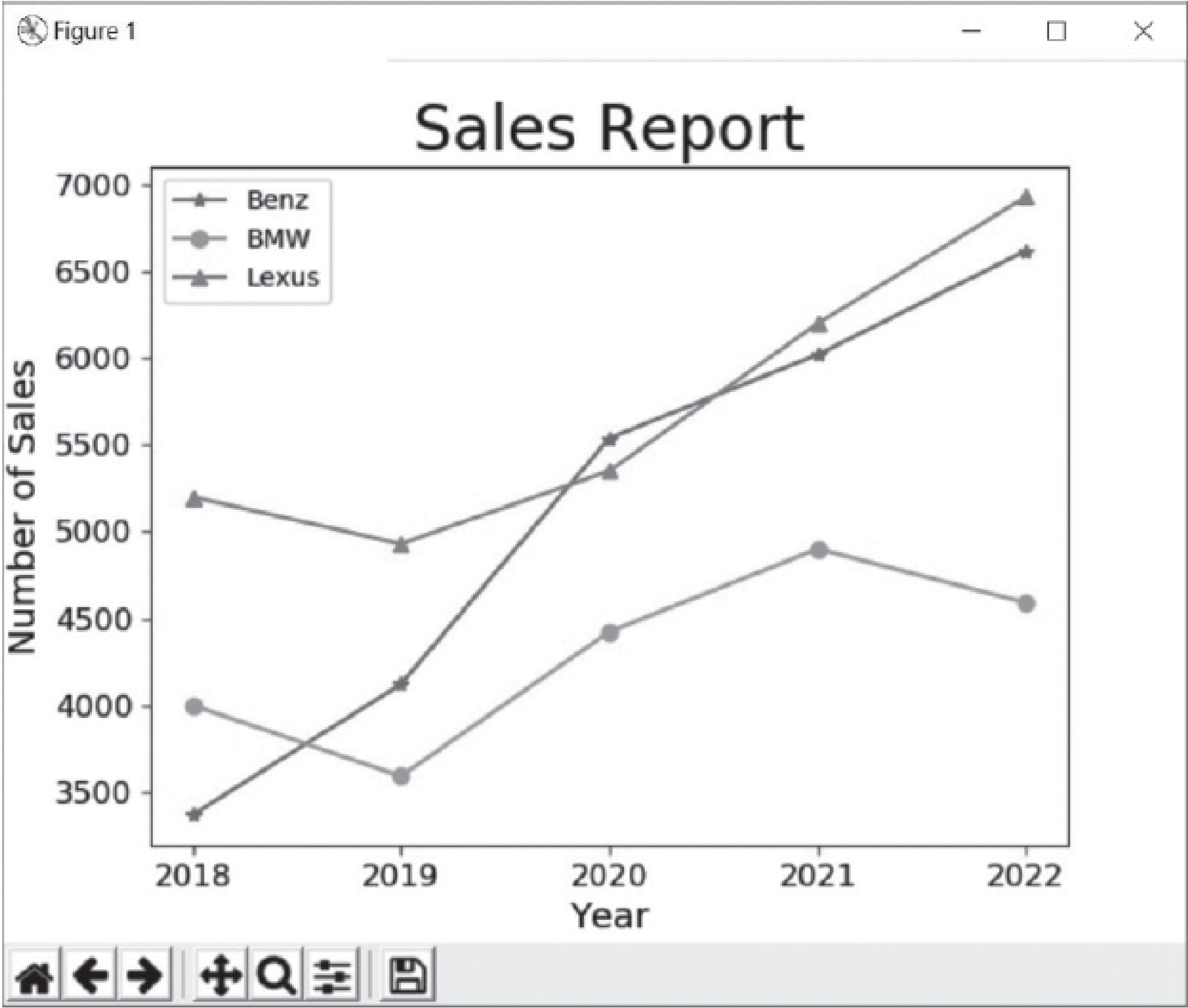
```
>>> stock2330["realtime"]["best_ask_price"]    #5个卖出金额
['221.00', '221.50', '222.00', '222.50', '223.00']
>>> stock2330["realtime"]["best_ask_volume"]   #5个卖出数量
['308', '4094', '1666', '474', '483']
```

习题

1. 请参考 ch20_12.py，增加 2021—2022 年数据如下。(20-1 节)

```
Benz      6020   6620
BMW 4900  4590
Lexus     6200   6930
```

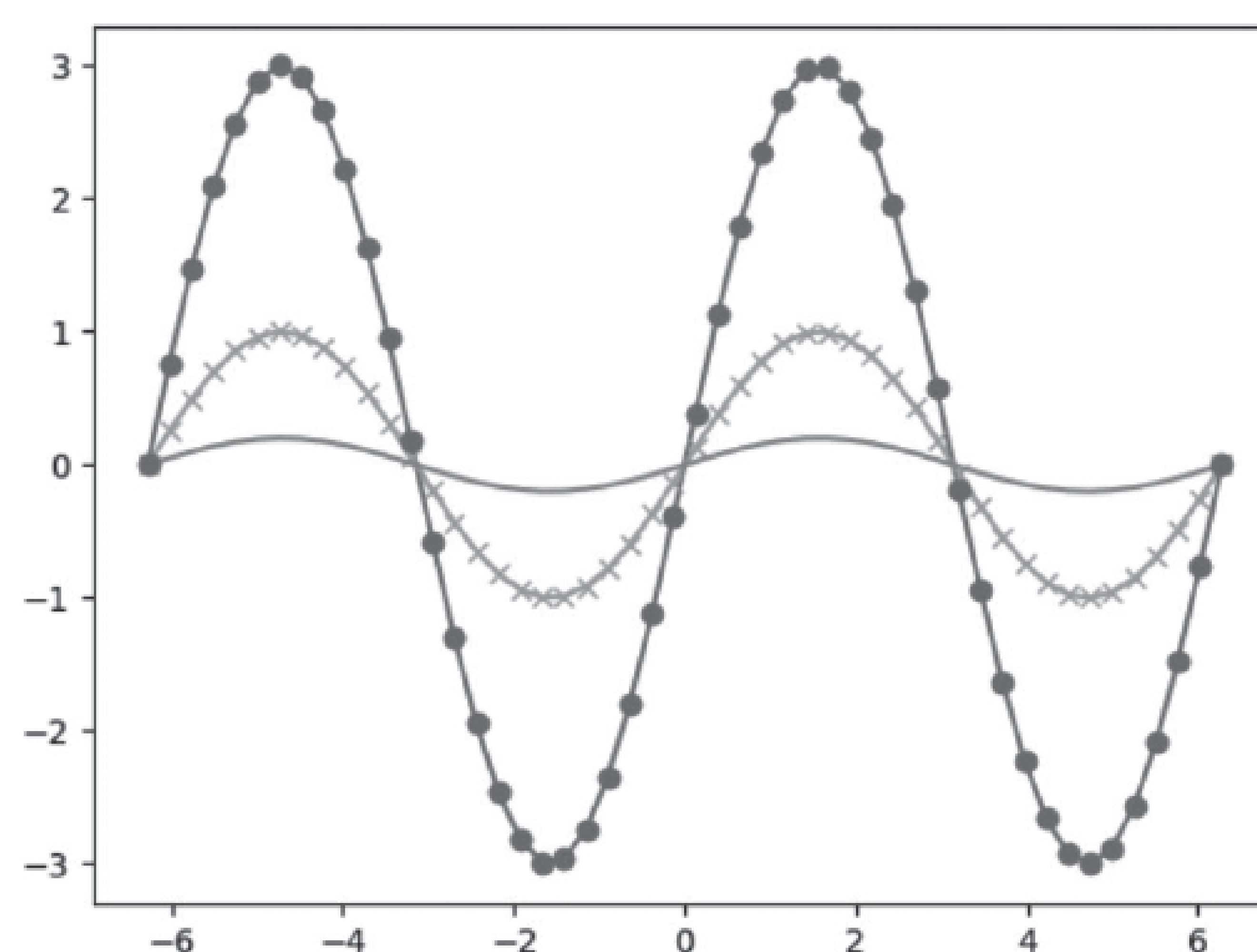
然后绘制图表。



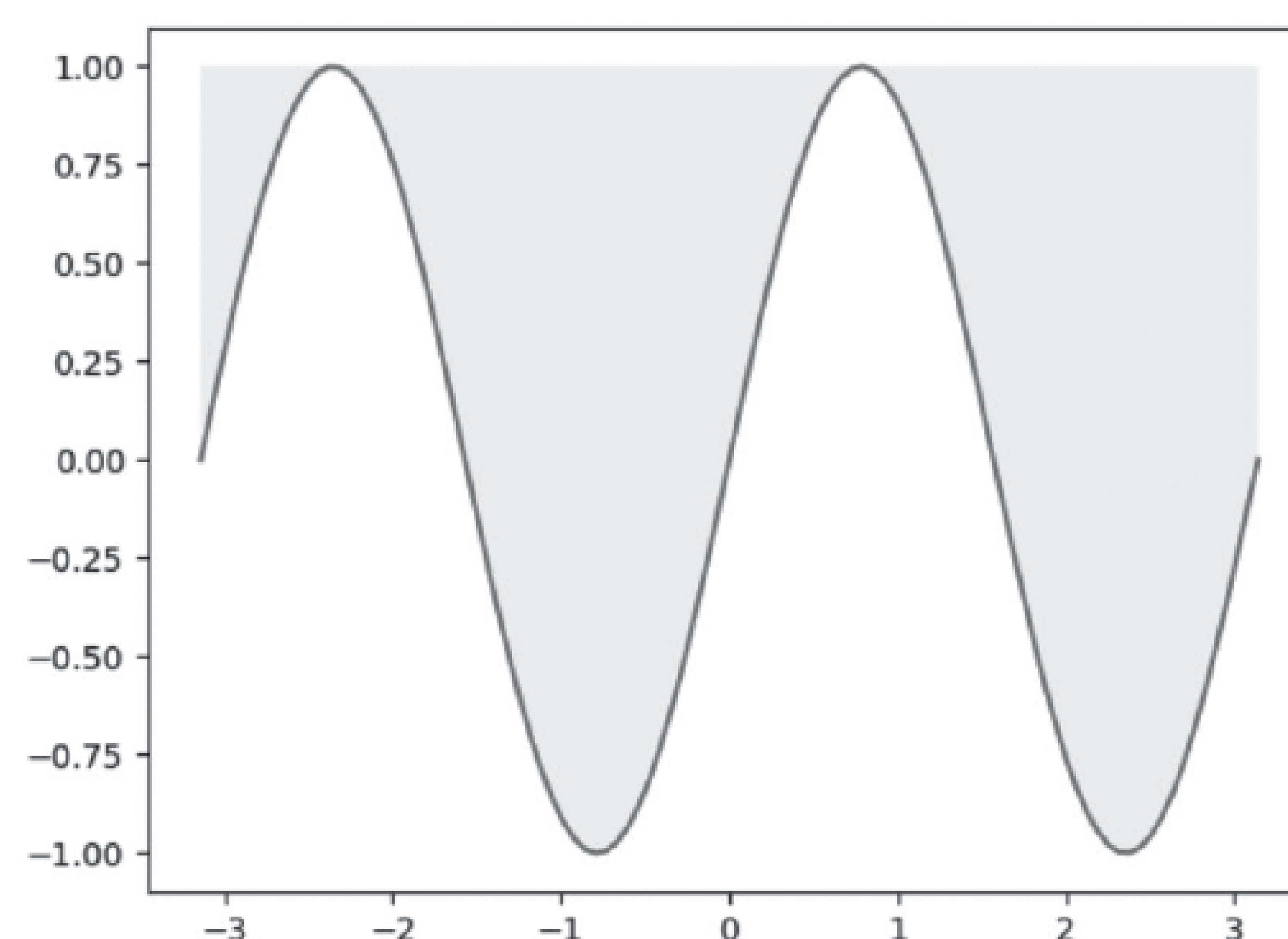
2. 请参考 ch20_19_1.py，但是将这 3 条线改为下列函数。(20-3 节)

```
f1 = 3 * np.sin(x)
f2 = np.sin(x)
f3 = 0.2.sin(x)
```

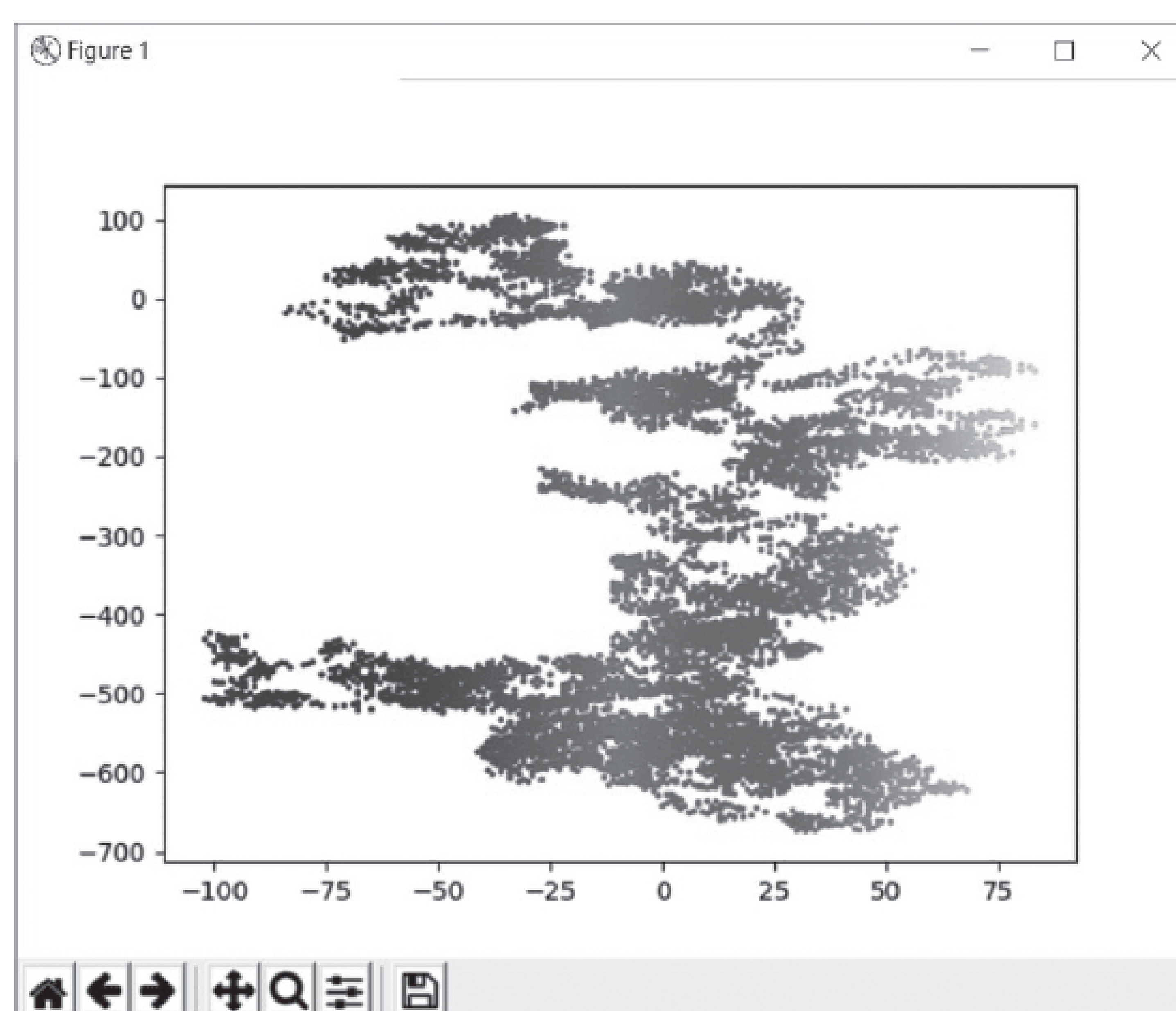
将线条点数改为 50，同时标注各点。f1 需用不同的默认颜色标注，这时需执行两次 plot( )。f2 则用相同线条颜色 'x' 标注。



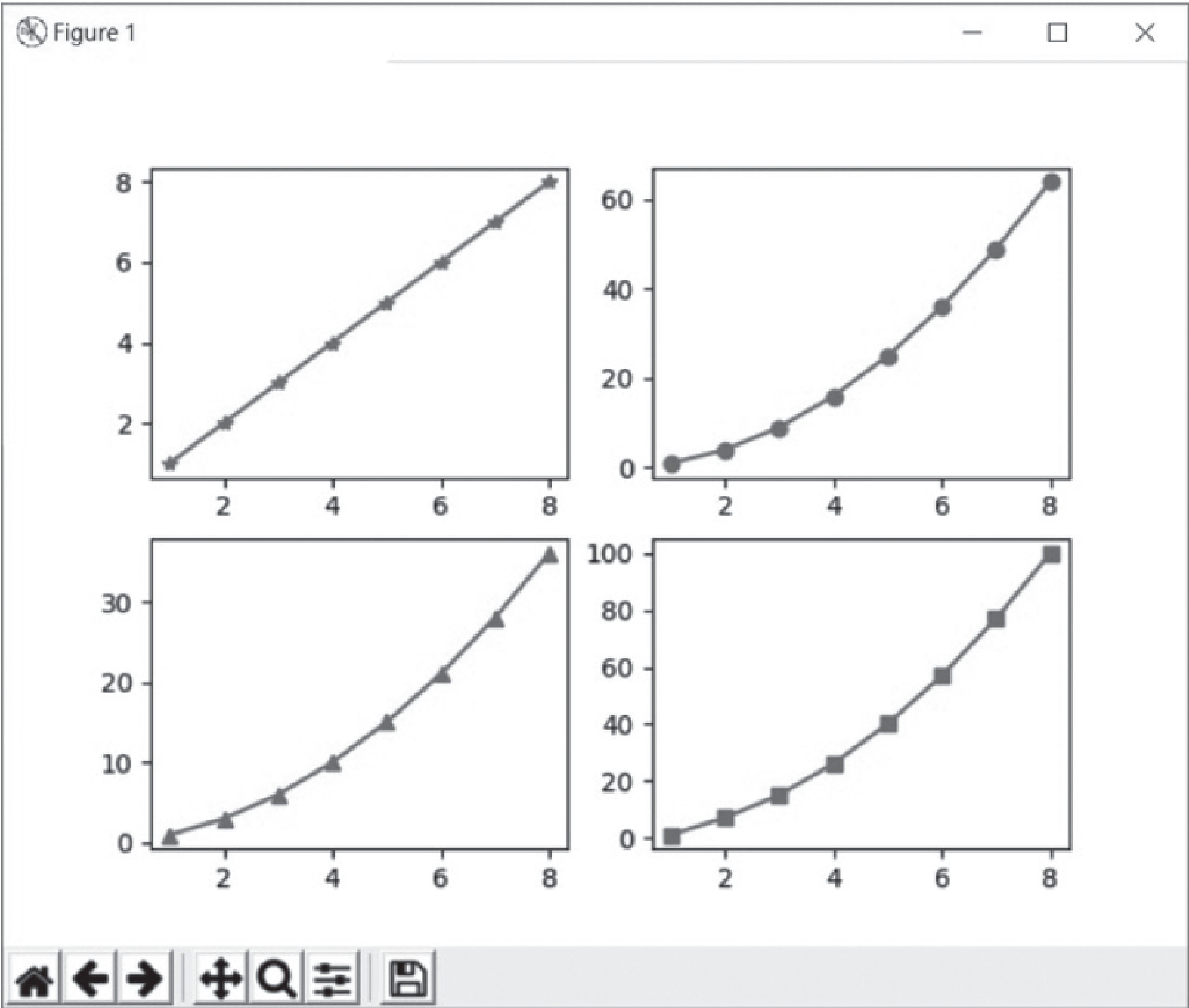
3. 请参考程序实例 ch20_20_2.py，将函数改为 sin(2x)，以默认的线条颜色绘制下列含填满区间的波形。(20-3 节)



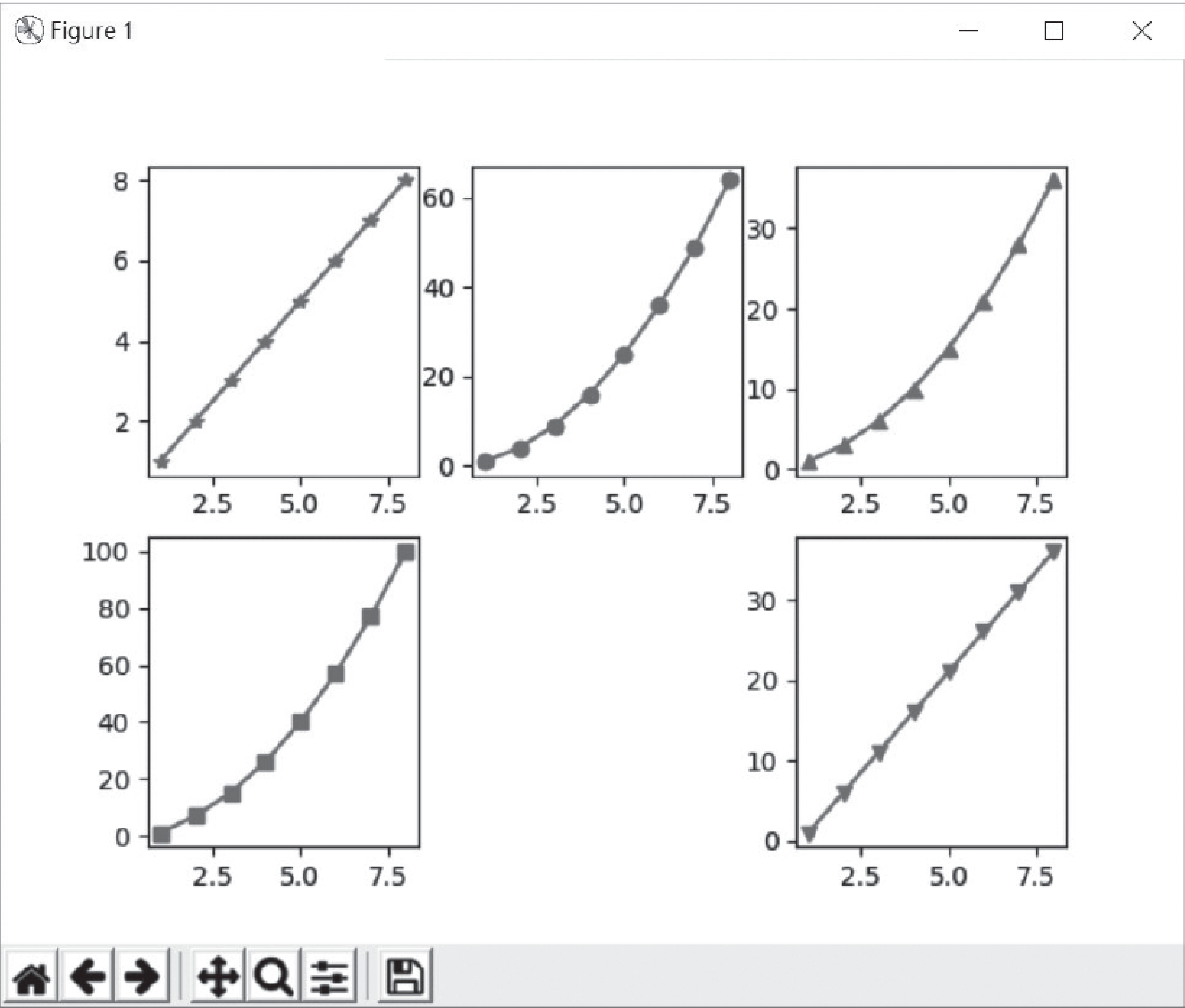
4. 请重新设计 ch20_22.py，将 x 轴移动方式改为 [−3, −2, −1, 1, 2, 3]，将 y 轴移动方式改为 [−5, −3, −1, 1, 3, 5]，然后列出结果。(20-4 节)



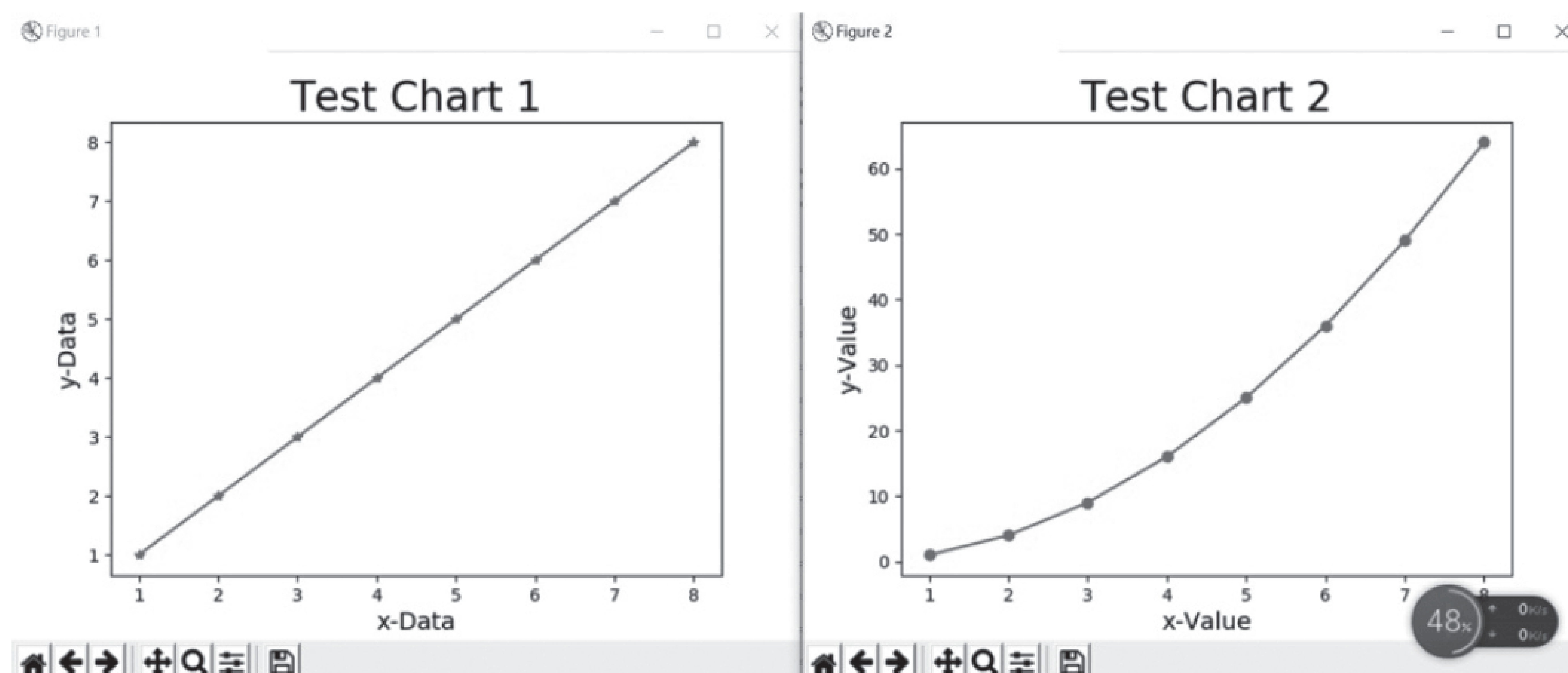
5. 请重新设计 ch20_9.py，将 4 组资料在 Figure1 内以 4 个子图方式显示。(20-5 节)



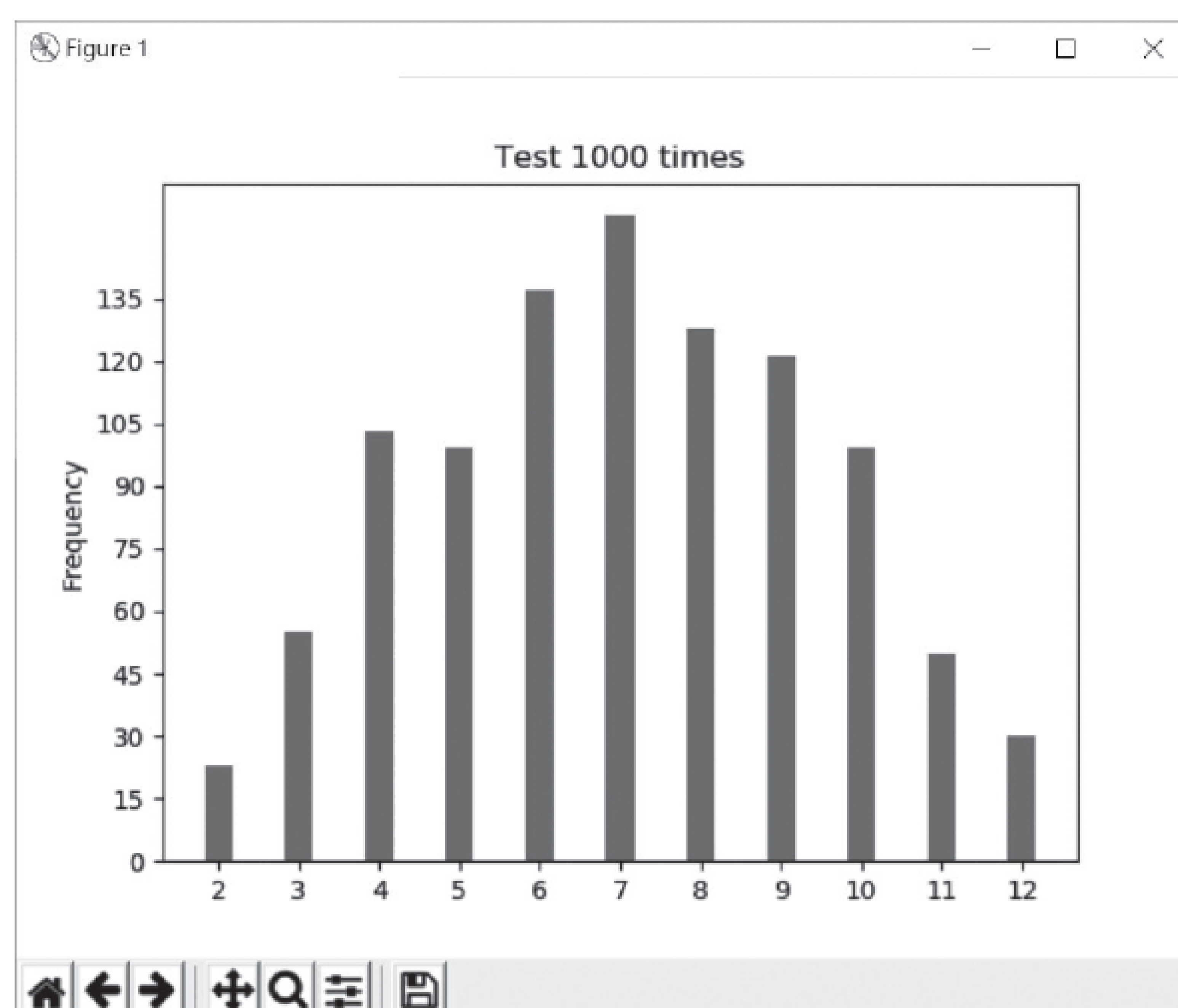
6. 请为 ch20_9.py 再增加 data5 数据，内容是 [1, 6, 11, 16, 21, 26, 31, 36]，然后将这 5 组数据绘制在 Figure 1 内分成 5 个子图，其中横向有 3 个子图，纵向有 2 个子图，第 5 个子图跳过，直接绘制在第 6 个图的位置，data5 数据的样式是反三角标记。(20-5 节)



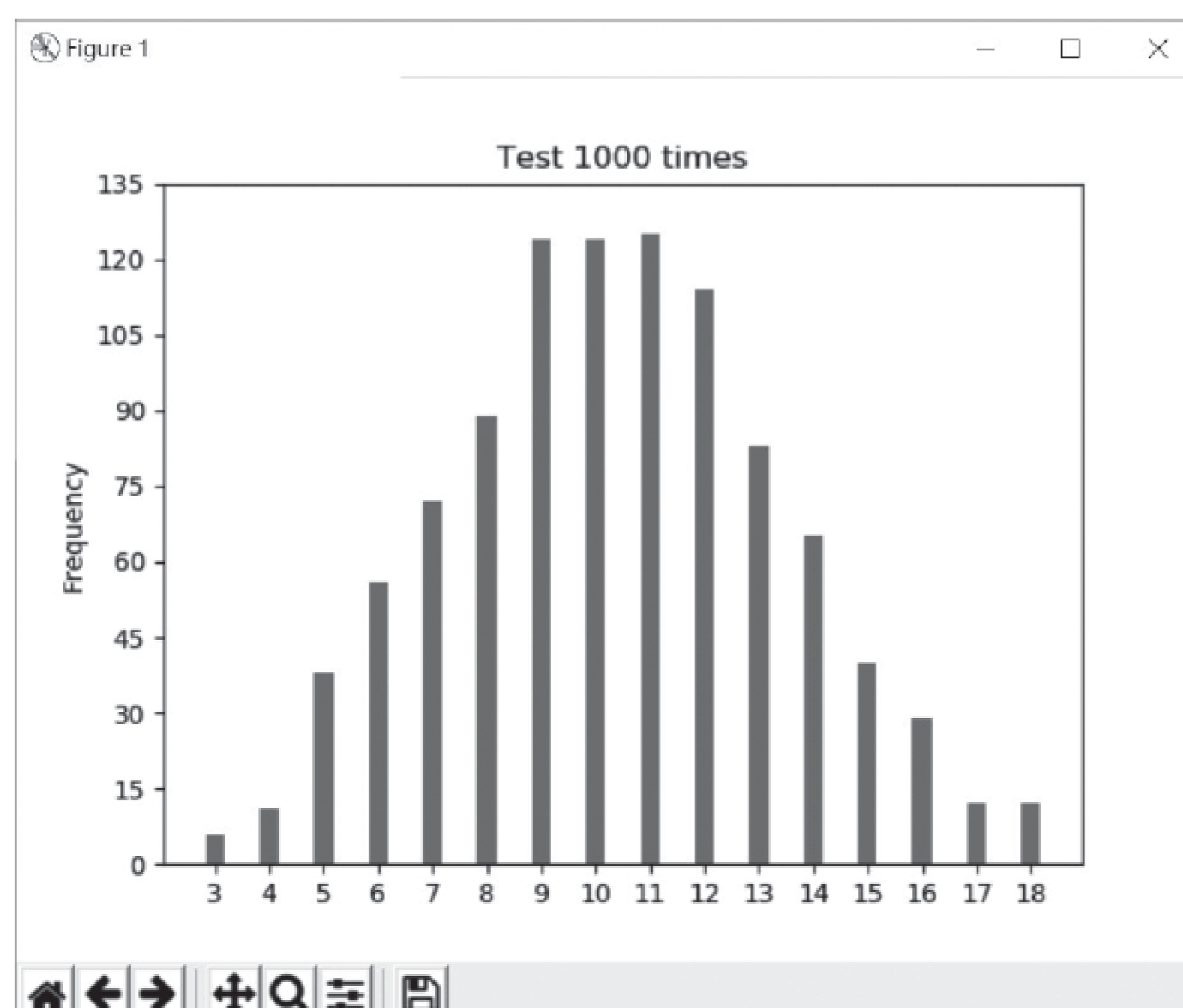
7. 扩充设计 ch20_24.py，为 Figure 1 增加标题 Test Chart1，x 轴和 y 轴标题分别是 x-Data 和 y-Data。(20-5 节)



8. 请读者将程序实例 ch20_28.py 处理成有两个骰子，所以可以计算 2 ～ 12 每个数字的出现次数，请测试 1000 次，以直方图表示。(20-6 节)



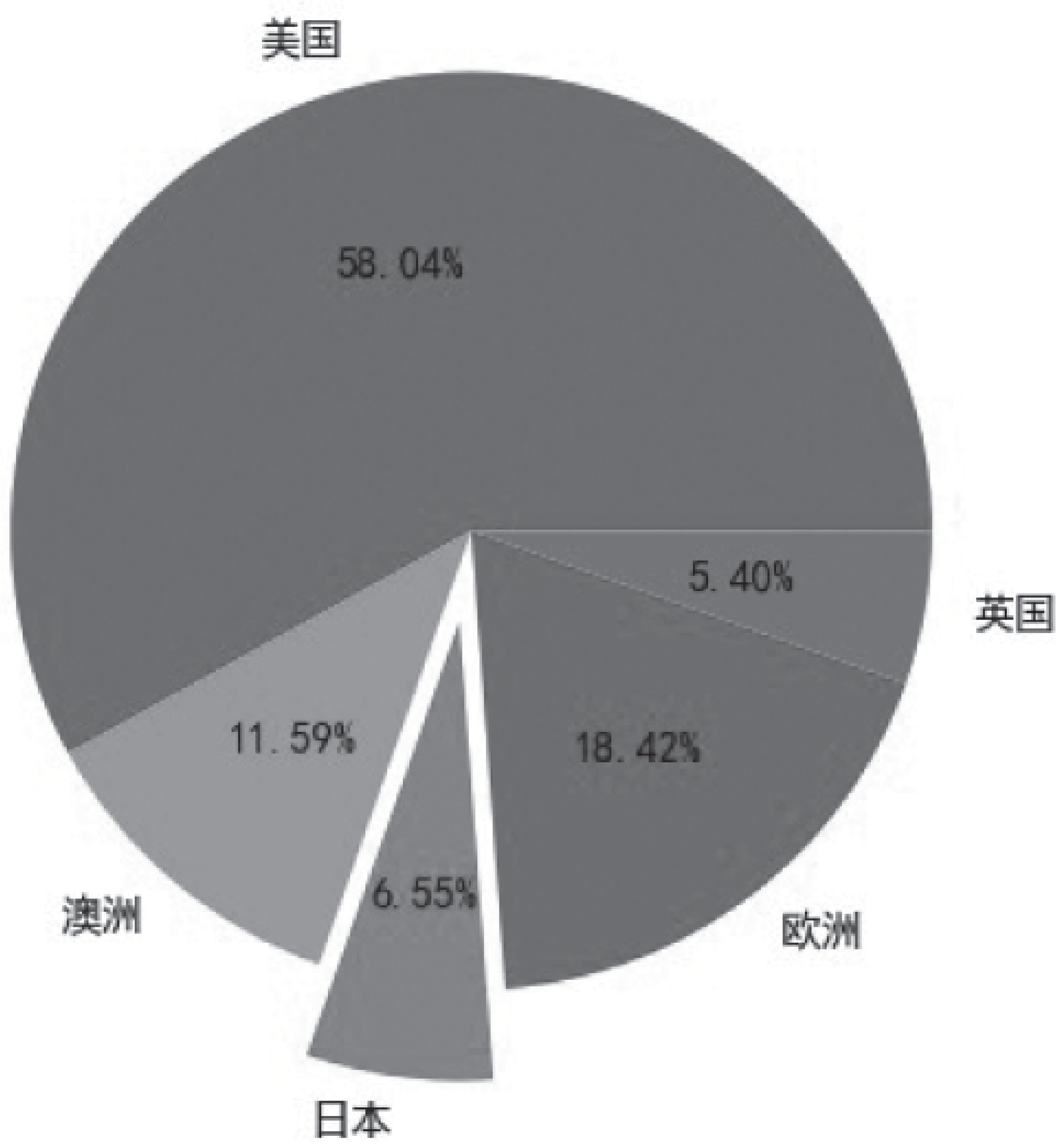
9. 请读者参考程序实例 ch20_28.py，在赌场最常见到的是用 3 个骰子，所以可以计算 3 ～ 18 每个数字的出现次数，请测试 1000 次，以直方图表示。(20-6 节)



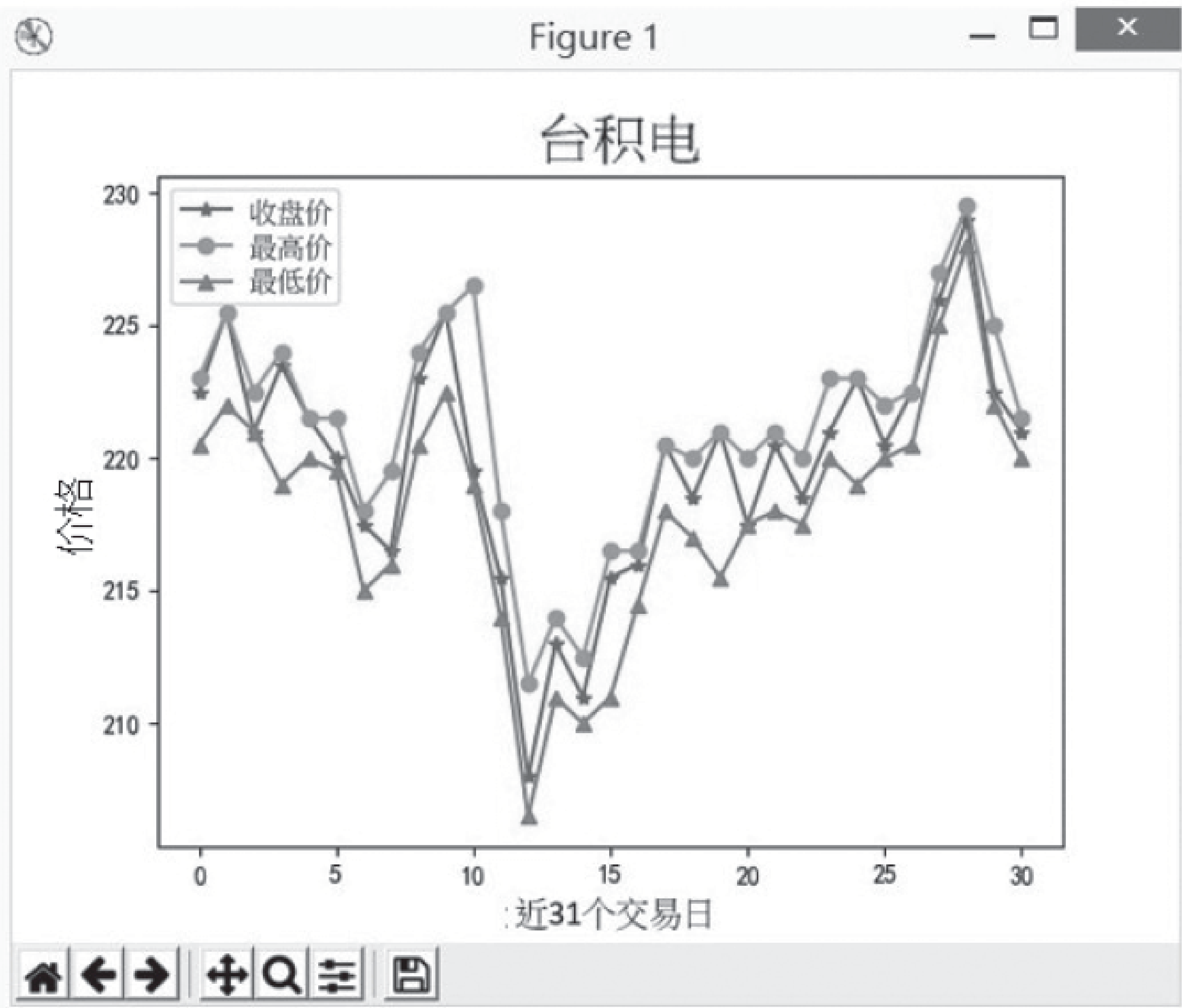
10. 下表是某年度中国台湾学生留学国外的统计数字表。(20-8 节)

美国	澳洲	日本	欧洲	英国
10543	2105	1190	3346	980

请绘制圆饼图，并将日本区块分离出来。



11. 请扩充程序实例 20_32.py，列出近 31 天台积电收盘价、最高价、最低价的折线图，同时需加上图例、x 轴和 y 轴的标题。(20-9 节)



12. 请设计 3 家股票公司近 31 天股票收盘价的折线图，同时需加上图例、x 轴与 y 轴的标题。(20-9 节)

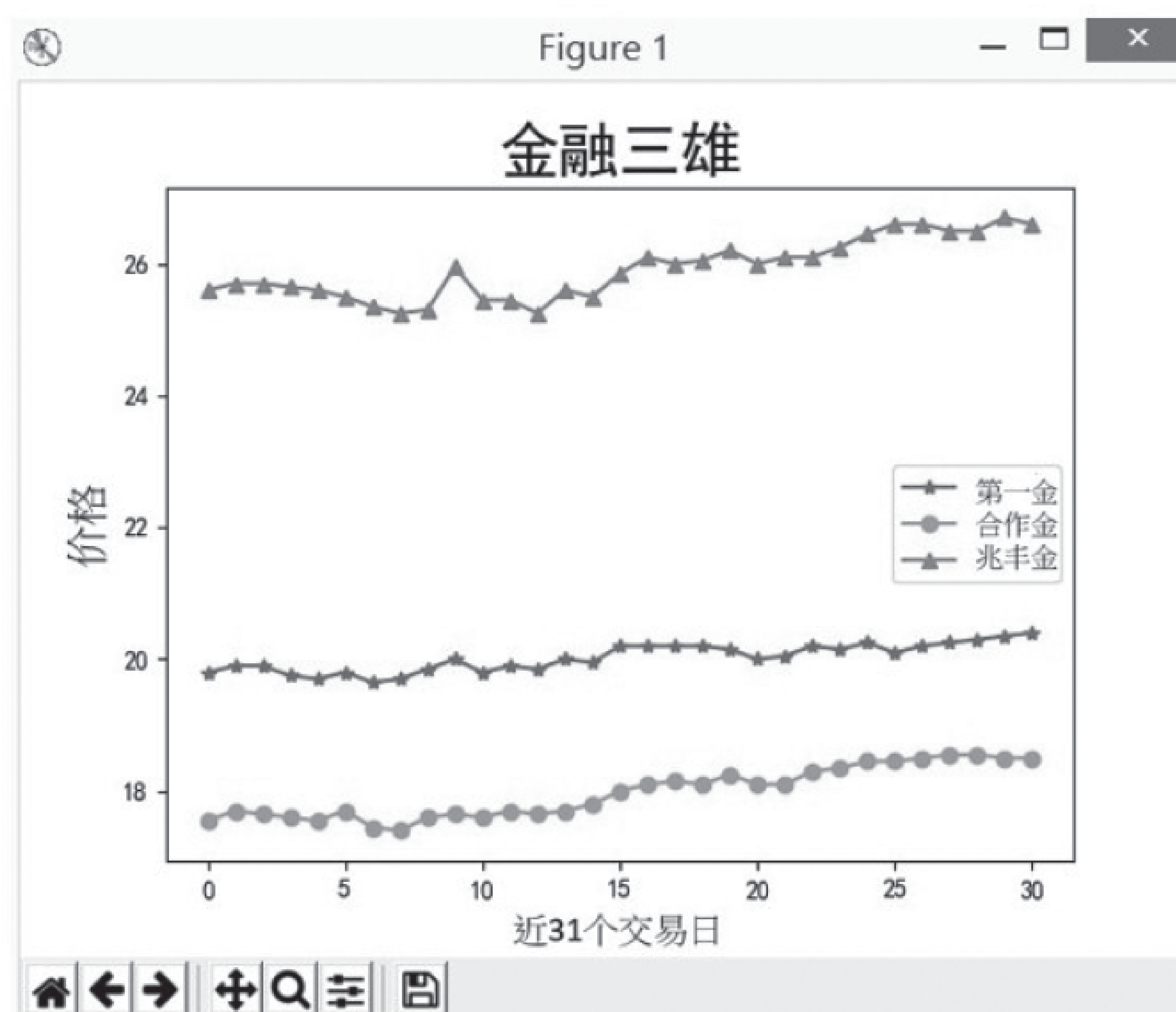


Figure 1
金融三雄
价格
近31个交易日
第一金
合作金
兆丰金
26
24
22
20
18
0
5
10
15
20
25
30

21

第 2 1 章

JSON 资料

本章摘要

JSON 是一种数据格式，由美国程序设计师 Douglas Crockford 创建。JSON 全名是 JavaScript Object Notation，由此我们可以推敲出 JSON 最初是为 JavaScript 开发的。这种数据格式由于简单好用，被大量应用在 Web 开发与大数据数据库（NoSQL），现在已成为一种著名的数据格式，Python 与许多程序语言都可以支持。因此，我们使用 Python 设计程序时，可以将数据以 JSON 格式储存，方便与使用其他程序语言的设计师分享。**注**：JSON 文件可以用记事本打开。

在 Python 中需使用 import json 导入 JSON 模块。

21-1 认识 JSON 数据格式

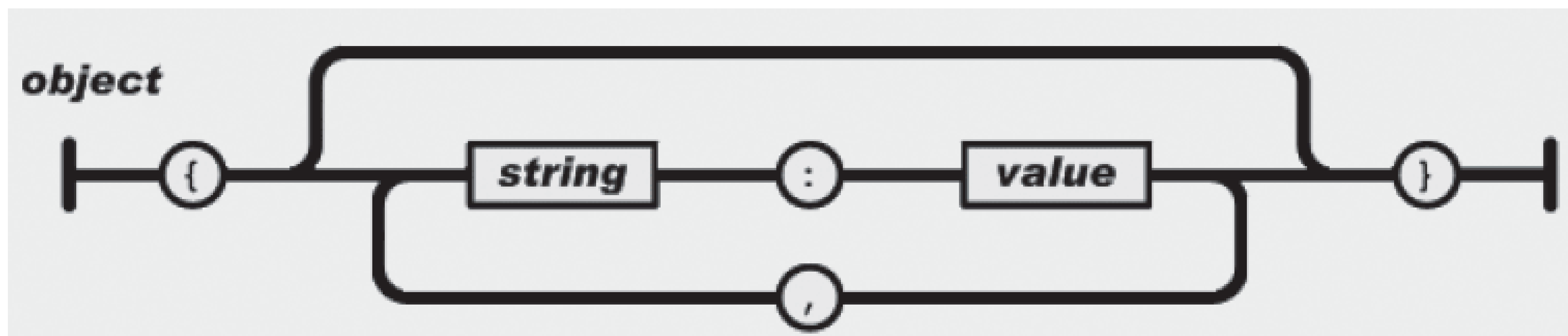
JSON 的数据格式有两种，分别是：

对象（object）：一般用大括号 { } 表示。

数组（array）：一般用中括号 [ ] 表示。

21-1-1 对象 (object)

在 JSON 中对象就是用“键 : 值（key:value）”的方式配对储存，对象内容用左大括号“{”开始，右大括号“}”结束，键（key）和值（value）用“:”间隔，每一组“键 : 值”间以逗号“,”隔开，以下是取材自 json.org 的官方说明图。



在 JSON 格式中，键（key）是一个字符串（string）。值可以是数值（number）、字符串（string）、布尔值（bool）、数组（array）或是 null 值。

例如，下列是对象的实例。

```
{"Name":"Hung", "Age":25}
```

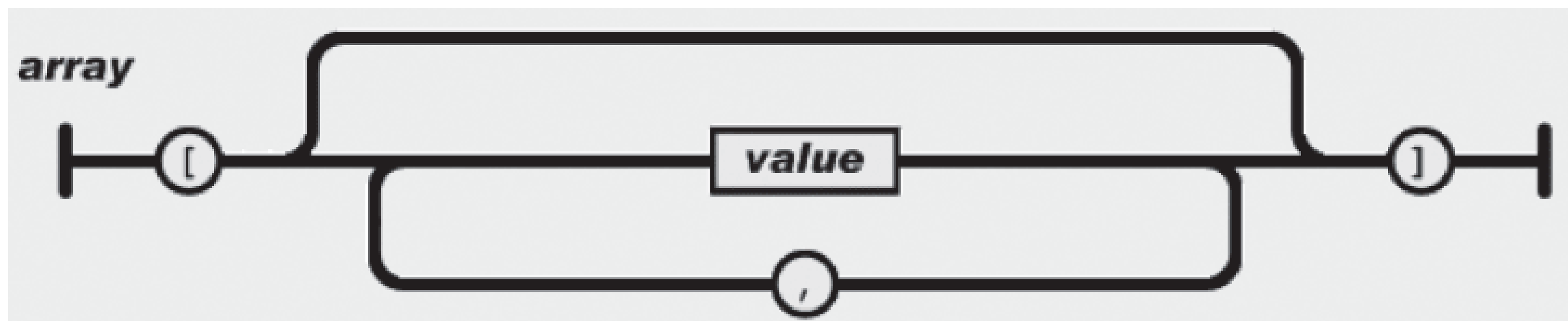
使用 JSON 时需留意，键（key）必须是文字，例如下列是错误的实例。

```
{"Name":"Hung", 25:"Key"}
```

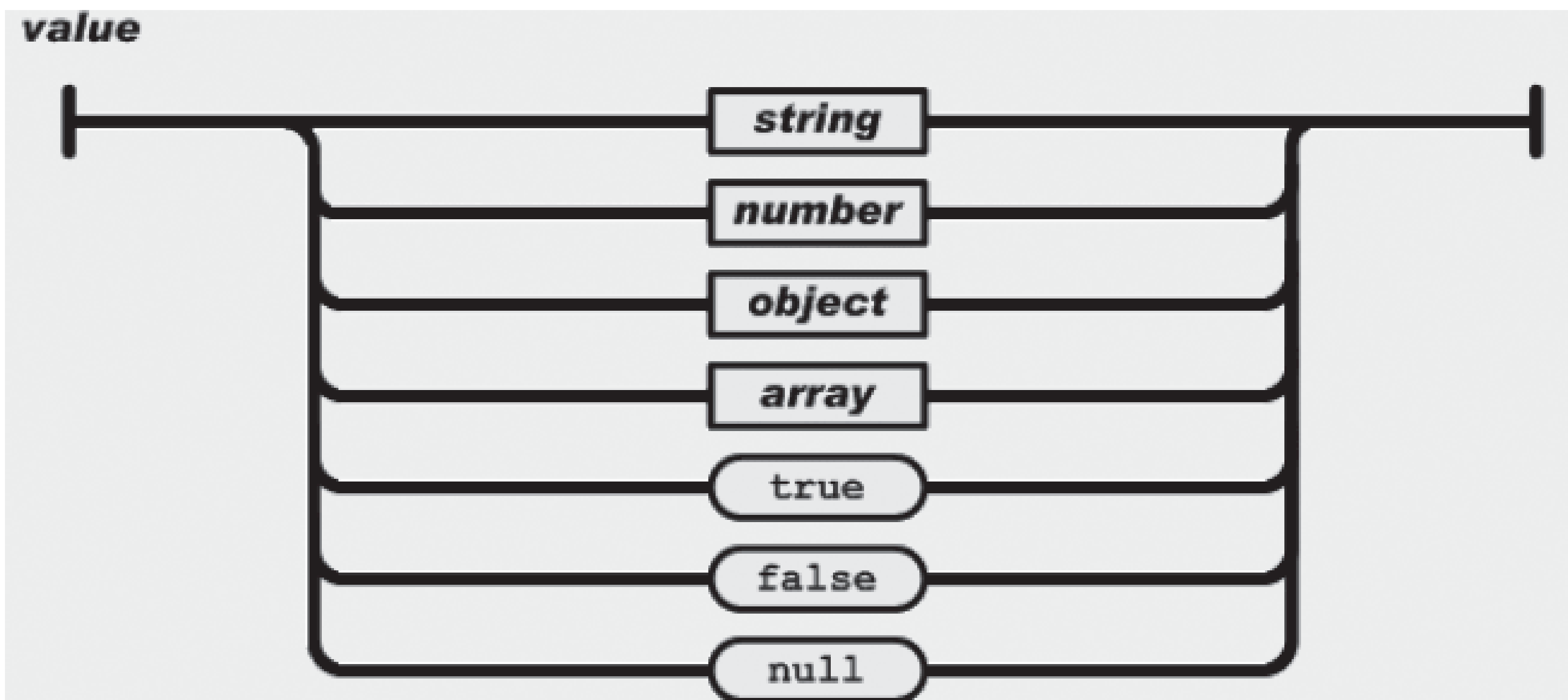
在 JSON 格式中字符串需用双引号，同时在 JSON 文件内不可以有注释。

21-1-2 数组 (array)

数组基本上是由一系列的值（value）所组成，用左中括号“[”开始，右中括号“]”结束，各值之间用逗号“,”隔开。以下是取材自 json.org 的官方说明图。



数组的值可以是数值（number）、字符串（string）、布尔值（bool）、数组（array）或是 null 值。



21-1-3 JSON 数据存在方式

前文所述是 JSON 的数据格式定义，但是在 Python 中它存在的方式是字符串（string）。

'JSON 数据'　　　　　　　　　# 可参考程序实例 ch21_1.py 的第 3 笔输出

使用 JSON 模块执行将 Python 数据转成 JSON 字符串类型数据或是 JSON 文件应使用不同方法，下列 21-2 和 21-3 节将分别说明。

21-2 将 Python 应用在 JSON 字符串形式数据

本节主要说明 JSON 数据以字符串形式存在时的应用。

21-2-1 使用 dumps( ) 将 Python 数据转成 JSON 格式

在 JSON 模块内有 dumps( )，可以将 Python 数据转成 JSON 字符串格式，下列是转化对照表。

Python 资料	JSON 资料
dict	object
list, tuple	array
str, unicode	string
int, float, long	number
True	true
False	false
None	null

程序实例 ch21_1.py：将 Python 的列表与元组数据转成 JSON 的数组数据。

```
1  # ch21_1.py
2  import json
3
4  listNumbers = [5, 10, 20, 1]              # 列表数据
5  tupleNumbers = (1, 5, 10, 9)              # 元组资料
6  jsonData1 = json.dumps(listNumbers)       # 将列表数据转成json数据
7  jsonData2 = json.dumps(tupleNumbers)      # 将列表数据转成json数据
8  print("列表转换成json的数组", jsonData1)
9  print("元组转换成json的数组", jsonData2)
10 print("json数组在Python的数据类型 ", type(jsonData1))
```

执行结果

```
==================== RESTART: D:\Python\ch21\ch21_1.py ====================
列表转换成json的数组 [5, 10, 20, 1]
元组转换成json的数组 [1, 5, 10, 9]
json数组在Python的数据类型  <class 'str'>
```

特别需要留意的是，上述笔者在第 10 行打印最终 JSON 数组在 Python 的数据类型时，结果是以字符串方式存在。若以 JSONData1 为例，从上述执行结果我们可以了解，在 Python 内它的数据如下：

‘[5, 10, 20, 1]’

程序实例 ch21_2.py：将 Python 由字典元素所组成的列表转成 JSON 数组，转换后原先字典元素变为 JSON 的对象。

```
1  # ch21_2.py
2  import json
3
4  listObj = [{'Name':'Peter', 'Age':25, 'Gender':'M'}]    # 列表数据元素是字典
5  jsonData = json.dumps(listObj)                          # 将列表数据转成json数据
6  print("列表转换成json的数组", jsonData)
7  print("json数组在Python的数据类型 ", type(jsonData))
```

执行结果

```
==================== RESTART: D:\Python\ch21\ch21_2.py ====================
列表转换成json的数组 [{"Name": "Peter", "Age": 25, "Gender": "M"}]
json数组在Python的数据类型  <class 'str'>
```

读者应留意，JSON 对象的字符串是用双引号。

21-2-2　dumps( ) 的 sort_keys 参数

Python 的字典是无序的数据，使用 dumps( ) 将 Python 数据转成 JSON 对象时，增加使用 sort_keys=True，则可以转成 JSON 格式的对象排序。

程序实例 ch21_3.py：将字典转成 JSON 格式的对象，分别是未使用排序与使用排序。最后将未使用排序与使用排序的对象做比较看是否相同，得到的结果是不同。

```
1  # ch21_3.py
2  import json
3
4  dictObj = {'b':80, 'a':25, 'c':60}                  # 字典
5  jsonObj1 = json.dumps(dictObj)                      # 未排序将字典转成json对象
6  jsonObj2 = json.dumps(dictObj, sort_keys=True)      # 有排序将字典转成json对象
7  print("未用排序将字典转换成json的对象", jsonObj1)
8  print("使用排序将字典转换成json的对象", jsonObj2)
9  print("有排序与未排序对象是否相同     ", jsonObj1 == jsonObj2 )
10 print("json物件在Python的数据类型 ", type(jsonObj1))
```

执行结果

```
==================== RESTART: D:\Python\ch21\ch21_3.py ====================
未用排序将字典转换成json的对象 {"b": 80, "a": 25, "c": 60}
使用排序将字典转换成json的对象 {"a": 25, "b": 80, "c": 60}
有排序与未排序对象是否相同      False
json物件在Python的数据类型  <class 'str'>
```

从上述执行结果可知 JSON 对象在 Python 的存放方式也是字符串。

21-2-3 dumps( ) 的 indent 参数

从 ch21_3.py 的执行结果可以看到数据不太容易阅读，特别是资料如果更多，在将 Python 的字典数据转成 JSON 格式的对象时，可以加上 indent 设置缩排 JSON 对象的键 - 值，让 JSON 对象可以更容易显示。

程序实例 ch21_4.py：将 Python 的字典转成 JSON 格式对象时，设置缩排 4 个字符宽度。

```
1  # ch21_4.py
2  import json
3
4  dictObj = {'b':80, 'a':25, 'c':60}                    # 字典
5  jsonObj = json.dumps(dictObj, sort_keys=True, indent=4) # 用内缩呈现json对象
6  print(jsonObj)
```

执行结果

```
==================== RESTART: D:\Python\ch21\ch21_4.py ====================
{
    "a": 25,
    "b": 80,
    "c": 60
}
```

21-2-4 使用 loads( ) 将 JSON 格式数据转成 Python 数据

在 JSON 模块内有 loads( )，可以将 JSON 格式数据转成 Python 数据，下列是转化对照表。

JSON 资料	Python 资料
object	dict
array	list
string	unicode
number(int)	int, long
Number(real)	float
true	True
false	False
null	None

程序实例 ch21_5.py：将 JSON 的对象数据转成 Python 数据，需留意在建立 JSON 数据时，需加上引号，因为 JSON 数据在 Python 内是以字符串形式存在。

```
1  # ch21_5.py
2  import json
3
4  jsonObj = '{"b":80, "a":25, "c":60}'    # json物件
5  dictObj = json.loads(jsonObj)           # 转成Python物件
6  print(dictObj)
7  print(type(dictObj))
```

执行结果

```
==================== RESTART: D:\Python\ch21\ch21_5.py ====================
{'b': 80, 'a': 25, 'c': 60}
<class 'dict'>
```

从上述可以看到 JSON 对象转成 Python 数据时的数据类型。

21-2-5　一个 JSON 文件只能放一个 JSON 对象

有一点要注意的是一个 JSON 文件只能放一个 JSON 对象，例如下列语法是无效的：

```
{"Japan":"Tokyo"}
{"China":"Beijing"}
```

如果要放多个 JSON 对象，可以用一个父 JSON 对象处理，上述可以更改成下列方式：

```
{"Asia":
[ {"Japan":"Tokyo"},
 {"China":"Beijing"} ]
}
```

Asia 是父 JSON，可以将“国家 : 首都”JSON 对象保存在数组中，未来用 Asia 存取此 JSON 资料。实际上这是一般 JSON 文件的配置方式，目前大部分网站的内部资料，就是用这种方式处理。

程序实例 ch21_5_1.py：建立一个父 JSON 对象，此父 JSON 对象内有 2 个子 JSON 对象。

```
1   # ch21_5_1.py
2   import json
3
4   obj = '{"Asia":[{"Japan":"Tokyo"},{"China":"Beijing"}]}'
5   json_obj = json.loads(obj)
6   print(json_obj)
7   print(json_obj["Asia"])
8   print(json_obj["Asia"][0])
9   print(json_obj["Asia"][1])
10  print(json_obj["Asia"][0]["Japan"])
11  print(json_obj["Asia"][1]["China"])
```

执行结果

```
==================== RESTART: D:/Python/ch21/ch21_5_1.py ====================
{'Asia': [{'Japan': 'Tokyo'}, {'China': 'Beijing'}]}
[{'Japan': 'Tokyo'}, {'China': 'Beijing'}]
{'Japan': 'Tokyo'}
{'China': 'Beijing'}
Tokyo
Beijing
```

上述程序可以执行，但是最大的缺点是第 4 行不容易阅读，此时我们可以用程序实例 ch21_5_2.py 中的方式改良。

程序实例 ch21_5_2.py：改良建立 JSON 数据的方法，让程序比较容易阅读，本程序使用 4 ～ 7 行改良原先的第 4 行。读者须留意，4 ～ 6 行每行末端应加上“\”，表示这是一个字符串。

```
1  # ch21_5_2.py
2  import json
3
4  obj = '{"Asia":\
5          [{"Japan":"Tokyo"},\
6           {"China":"Beijing"}]\
7         }'
8  json_obj = json.loads(obj)
9  print(json_obj)
10 print(json_obj["Asia"])
11 print(json_obj["Asia"][0])
12 print(json_obj["Asia"][1])
13 print(json_obj["Asia"][0]["Japan"])
14 print(json_obj["Asia"][1]["China"])
```

执行结果

与 ch21_5_1.py 相同。

21-3 将 Python 应用在 JSON 文件

我们在设计程序时，更重要的是将 Python 的资料以 JSON 格式储存，未来可以供其他不同的程序语言读取。或是使用 Python 读取其他语言以 JSON 格式储存的数据。

21-3-1 使用 dump( ) 将 Python 数据转成 JSON 文件

在 JSON 模块内有 dump( )，可以将 Python 数据转成 JSON 文件格式，这个文件格式的扩展名是 json。下列将直接以程序实例解说 dump( ) 的用法。

程序实例 ch21_6.py：将一个字典数据使用 JSON 格式存储在 out21_6.JSON 文件内。在这个程序实例中，dump( ) 方法的第一个参数是要存储成 JSON 格式的数据，第二个参数是要存储的文件对象。

```
1  # ch21_6.py
2  import json
3
4  dictObj = {'b':80, 'a':25, 'c':60}
5  fn = 'out21_6.json'
6  with open(fn, 'w') as fnObj:
7      json.dump(dictObj, fnObj)
```

执行结果

在目前工作文件夹可以新增 JSON 文件，文件名是 out21_6.json。如果用记事本打开，可以得到下列结果。

```
{"b": 80, "a": 25, "c": 60}
```

21-3-2　使用 load( ) 读取 JSON 文件

在 JSON 模块内有 load( ) 可以读取 JSON 文件，读完后这个 JSON 文件将被转换成 Python 的数据格式，下列将直接以程序实例解说 load( ) 的用法。

程序实例 ch21_7.py：读取 JSON 文件 out21_6.json，同时列出结果。

```
1  # ch21_7.py
2  import json
3
4  fn = 'out21_6.json'
5  with open(fn, 'r') as fnObj:
6      data = json.load(fnObj)
7
8  print(data)
9  print(type(data))
```

执行结果

```
===================== RESTART: D:\Python\ch21\ch21_7.py =====================
{'b': 80, 'a': 25, 'c': 60}
<class 'dict'>
```

21-3-3　将中文字典数据转成 JSON 文件

如果想要存储的字典数据包含中文，使用上一小节的方式，将造成打开此 JSON 文件时，以 16 进位码值方式显示（\uxxxx）。如果以记事本打开，则会造成文件不易了解内容。

程序实例 ch21_9_1.py：建立串行，此串行的元素是中文字典数据，然后存储成 JSON 文件，文件名是 out21_9_1.json，最后以记事本打开此文件。

```
1  # ch21_9_1.py
2  import json
3
4  objlist = [{"日本":"Japan", "首都":"Tykyo"},
5             {"美国":"USA", "首都":"Washington"}]
6
7  fn = 'out21_9_1.json'
8  with open(fn, 'w') as fnObj:
9      json.dump(objlist, fnObj)
```

执行结果 下列是以记事本打开此文件的结果。

```
[{"\u65e5\u672c": "Japan", "\u9996\u90fd": "Tykyo"}, {"\u7f8e\u5dde": "USA", "\u9996\u90fd": "Washington"}]
```

如果我们想要顺利显示所储存的中文数据，在打开文件时，可以增加使用 encoding=utf-8 参数。同时在使用 json.dump( ) 时，增加 ensure_ascii=False，意义是中文以中文方式写入（utf-8 编码方式写入），如果没有或是 ensure_ascii 是 True 时，中文以 \uxxxx 格式写入。此外，我们一般会在 json.dump( ) 内增加 indent 参数，这是设置字典元素内缩字符数，常见是设为 indent=2。

程序实例 ch21_9_2.py：使用 utf-8 格式搭配 ensure_ascii=False 存储中文字典数据，同时设置 indent=2，请将结果存储至 out21_9_2.json。

```
1  # ch21_9_2.py
2  import json
3
4  objlist = [{"日本":"Japan", "首都":"Tykyo"},
5             {"美国":"USA", "首都":"Washington"}]
6
7  fn = 'out21_9_2.json'
8  with open(fn, 'w', encoding='utf-8') as fnObj:
9      json.dump(objlist, fnObj, indent=2, ensure_ascii=False)
```

执行结果 下列是使用记事本打开的结果。

```
[
  {
    "日本": "Japan",
    "首都": "Tykyo"
  },
  {
    "美国": "USA",
    "首都": "Washington"
  }
]
```

21-4 简单的 JSON 文件应用

程序实例 ch21_8.py：程序执行时会要求输入账号，然后列出所输入账号并打印“欢迎使用本系统”。

```
1  # ch21_8.py
2  import json
3
4  fn = 'login.json'
5  login = input("请输入账号 : ")
6  with open(fn, 'w') as fnObj:
7      json.dump(login, fnObj)
8      print("%s! 欢迎使用本系统! " % login)
```

执行结果

```
==================== RESTART: D:\Python\ch21\ch21_8.py ====================
请输入账号 : Peter
Peter! 欢迎使用本系统!
```

上述程序同时会将所输入的账号存入 login.json 文件内。

程序实例 ch21_9.py：读取 login.json 的数据，同时输出“欢迎回来使用本系统”。

```
1  # ch21_9.py
2  import json
3
4  fn = 'login.json'
5  with open(fn, 'r') as fnObj:
6      login = json.load(fnObj)
7      print("%s! 欢迎回来使用本系统! " % login)
```

执行结果

```
===================== RESTART: D:\Python\ch21\ch21_9.py =====================
Peter! 欢迎回来使用本系统!
```

程序实例 ch21_10.py：下列程序基本上是 ch21_8.py 和 ch21_9.py 的组合，如果第一次登录会要求输入账号，然后将输入账号记录在 login21_10.json 文件内。如果不是第一次登录，会直接读取已经存在 login21_10.json 的账号，然后打印 “欢迎回来”。这个程序用第 7 行是否能正常读取 login21_10.json 的方式判断是否是第一次登录，如果这个文件不存在，表示是第一次登录，将执行第 8 行 except 至第 12 行的内容。如果这个文件已经存在，表示不是第一次登录，将执行第 13 行 else: 后面的内容。

```
1  # ch21_10.py
2  import json
3
4  fn = 'login21_10.json'
5  try:
6      with open(fn) as fnObj:
7          login = json.load(fnObj)
8  except Exception:
9      login = input("请输入账号 : ")
10     with open(fn, 'w') as fnObj:
11         json.dump(login, fnObj)
12         print("系统已经记录你的账号 ")
13 else:
14     print("%s 欢迎回来" % login)
```

执行结果

```
==================== RESTART: D:\Python\ch21\ch21_10.py ====================
请输入账号 : Peter
系统已经记录你的账号
>>>
==================== RESTART: D:\Python\ch21\ch21_10.py ====================
Peter 欢迎回来
>>>
```

21-5　人口数据的 JSON 文件

在本书 ch21 文件夹内有 populations.json 文件，这是一个非官方的 2000 年和 2010 年的人口统计数据。这一节笔者将一步一步讲解如何使用 JSON 数据文件。

21-5-1　认识人口统计的 JSON 文件

若是将这个文件用记事本打开，内容如下：


```
[{"Country Name": "World", "Country Code": "WLD", "Year": "2000", "Numbers": "6117806174.56156"}, {"Coun
bers": "65258.0"}, {"Country Name": "Andorra", "Country Code": "AND", "Year": "2010", "Numbers": "85216.0"
"Numbers": "108186.0"}, {"Country Name": "Australia", "Country Code": "AUS", "Year": "2000", "Numbers": "19
"2000", "Numbers": "129592417.0"}, {"Country Name": "Bangladesh", "Country Code": "BGD", "Year": "2010", "
umbers": "8850223.0"}, {"Country Name": "Bermuda", "Country Code": "BMU", "Year": "2000", "Numbers": "62
"2000", "Numbers": "174425502.0"}, {"Country Name": "Brazil", "Country Code": "BRA", "Year": "2010", "Numb
try Code": "KHM", "Year": "2010", "Numbers": "14139608.0"}, {"Country Name": "Cameroon", "Country Code":
13.0"}, {"Country Name": "Chad", "Country Code": "TCD", "Year": "2000", "Numbers": "8223089.0"}, {"Country I
55.0"}, {"Country Name": "Comoros", "Country Code": "COM", "Year": "2010", "Numbers": "735266.0"}, {"Coun
, "Year": "2010", "Numbers": "4418192.0"}, {"Country Name": "Cuba", "Country Code": "CUB", "Year": "2000", "
ode": "DJI", "Year": "2000", "Numbers": "732112.0"}, {"Country Name": "Djibouti", "Country Code": "DJI", "Year
ntry Name": "El Salvador", "Country Code": "SLV", "Year": "2010", "Numbers": "6193287.0"}, {"Country Name":
bers": "49157.0"}, {"Country Name": "Fiji", "Country Code": "FJI", "Year": "2000", "Numbers": "812309.0"}, {"Cou
"1297212.0"}, {"Country Name": "Gambia, The", "Country Code": "GMB", "Year": "2010", "Numbers": "1729998
```

在网络上任何一个号称是真实统计的 JSON 数据，在用记事本打开后，初看一定是复杂的。读者碰上这个问题首先不要慌，可以分析一下数据的共通性，这样有助于未来程序的规划与设计。从上图我们基本可以了解它的资料格式，这是一个列表，列表元素是字典，有些国家只有 2000 年的数据，有些国家只有 2010 年的数据，有些国家则同时有这两个年度的数据，每个字典内有 4 个“键 : 值”，如下所示：

```
{
    "Country Name":"World",
    "Country Code":"WLD",
    "Year":"2000",
    "Numbers":"6117806174.56156"
}
```

上述字段分别是国家名称（Country Name）、国家代码（Country Code）、年份（Year）和人口数（Numbers）。从上述文件我们注意到，人口数在我们日常生活理解中应该是整数，可是这个数据中是用字符串表达。另外，在非官方的统计数据中，难免会有错误，例如，上述全球人口统计（Country Name 为 World）的数据出现了小数点，这个皆须我们用程序处理。

程序实例 ch21_11.py：列出 populations.json 数据中各国的代码，以及列出 2000 年各国人口数据。

```
1  # ch21_11.py
2  import json
3
4  fn = 'populations.json'
5  with open(fn) as fnObj:
6      getDatas = json.load(fnObj)                     # 读json档案
7
8  for getData in getDatas:
9      if getData['Year'] == '2000':                   # 筛选2000年的数据
10         countryName = getData['Country Name']       # 国家名称
11         countryCode = getData['Country Code']       # 国家代码
12         population = int(float(getData['Numbers'])) # 人口数据
13         print('国家代码 =', countryCode,
14               '国家名称 =', countryName,
15               '人口数 =', population)
```

执行结果

```
===================== RESTART: D:\Python\ch21\ch21_11.py =====================
国家代码 = WLD 国家名称 = World 人口数 = 6117806174
国家代码 = AFG 国家名称 = Afghanistan 人口数 = 25951672
国家代码 = ALB 国家名称 = Albania 人口数 = 3072478
国家代码 = DZA 国家名称 = Algeria 人口数 = 30534041
国家代码 = ASM 国家名称 = American Samoa 人口数 = 57995
国家代码 = AND 国家名称 = Andorra 人口数 = 65258
国家代码 = AGO 国家名称 = Angola 人口数 = 13926705
国家代码 = ATG 国家名称 = Antigua and Barbuda 人口数 = 78536
国家代码 = ARG 国家名称 = Argentina 人口数 = 36931013
国家代码 = ARM 国家名称 = Armenia 人口数 = 3076653
国家代码 = ABW 国家名称 = Aruba 人口数 = 91031
国家代码 = AUS 国家名称 = Australia 人口数 = 19153581
```

上述重点是第 12 行，当我们碰上含有小数点的字符串时，须先将这个字符串转成浮点数，然后再将浮点数转成整数。

21-5-2 认识 pygal.maps.world 的国家代码信息

前一节 populations.json 中国家代码是 3 个英文字母，如果我们想要使用这个 JSON 数据绘制人口地图，需要配合使用 pygal.maps.world 模块。这个模块的国家代码是 2 个英文字母，所以需要将

populations.json 的国家代码转成 2 个英文字母。pygal.maps.world 模块内有 COUNTRIES 字典，在这个字典中可以找到相关国家与代码的列表。使用 pygal.maps.world 模块前需先安装此模块，如下所示：

```
pip install pygal_maps_world
```

程序实例 ch21_12.py：列出 pygal.maps.world 模块 COUNTRIES 字典的 2 个英文字符的国家代码与完整的国家名称列表。

```
1  # ch21_12.py
2  from pygal.maps.world import COUNTRIES
3
4  for countryCode in sorted(COUNTRIES.keys()):
5      print("国家代码 :", countryCode, "   国家名称 = ", COUNTRIES[countryCode])
```

执行结果

```
===================== RESTART: D:\Python\ch21\ch21_12.py =====================
国家代码 : ad   国家名称 =  Andorra
国家代码 : ae   国家名称 =  United Arab Emirates
国家代码 : af   国家名称 =  Afghanistan
国家代码 : al   国家名称 =  Albania
国家代码 : am   国家名称 =  Armenia
国家代码 : ao   国家名称 =  Angola
国家代码 : aq   国家名称 =  Antarctica
国家代码 : ar   国家名称 =  Argentina
国家代码 : at   国家名称 =  Austria
国家代码 : au   国家名称 =  Australia
```

接着让程序在输出 2 个字母的国家代码时，同时输出此国家名称，这个程序相当于是将 2 个不同来源的数据做配对。



程序实例 ch21_13.py：从 populations.json 中提取每个国家的名称信息，然后将每一个国家名称放入 getCountryCode( ) 方法中找寻相关的代码，如果有找到则输出相对应的国家代码，如果找不到则输出“名称不吻合”。

```
1  # ch21_13.py
2  import json
3  from pygal.maps.world import COUNTRIES
4
5  def getCountryCode(countryName):
6      '''输入国家名称回传国家代码'''
7      for dictCode, dictName in COUNTRIES.items():    # 查找国家与国家代码字典
8          if dictName == countryName:
9              return dictCode                         # 如果找到则返回国家代码
10     return None                                     # 找不到则返回None
11
12 fn = 'populations.json'
13 with open(fn) as fnObj:
14     getDatas = json.load(fnObj)                     # 读取人口数据json档案
15
16 for getData in getDatas:
17     if getData['Year'] == '2000':                   # 筛选2000年的数据
18         countryName = getData['Country Name']       # 国家名称
19         countryCode = getCountryCode(countryName)
20         population = int(float(getData['Numbers'])) # 人口数
21         if countryCode != None:
22             print(countryCode, ":", population)     # 国家名称相符
23         else:
24             print(countryName," 名称不吻合:")        # 国家名称不吻合
```

执行结果

```
===================== RESTART: D:\Python\ch21\ch21_13.py =====================
World  名称不吻合:
af : 25951672
al : 3072478
dz : 30534041
American Samoa  名称不吻合:
ad : 65258
ao : 13926705
Antigua and Barbuda  名称不吻合:
```

上述会有不吻合输出是因为这是 2 个不同单位的数据，例如，有的数据在 populations.json 中有记录，在 pygal.maps.world 模块的 COUNTRIES 字典中则没有这个纪录。

22

第 2 2 章

使用 Python 处理 CSV 文件

本章摘要

22-1 建立一个 CSV 文件
22-2 用记事本打开 CSV 文件
22-3 CSV 模块
22-4 读取 CSV 文件
22-5 写入 CSV 文件
22-6 专题——使用 CSV 文件绘制气象图表

CSV 是一个缩写，它的英文全名是 Comma-Separated Values，由字面意义可以理解为“逗号分隔值”，当然逗号是主要数据字段间的分隔值，不过目前也有非逗号的分隔值。这是一个纯文本格式的文件，没有图片，也不用考虑字形、大小、颜色等。

简单地说，CSV 数据是指同一行（row）的资料彼此用逗号（或其他符号）隔开，同时每一行数据是一笔（record）数据，几乎所有电子表格与数据库文件均支持这个文件格式。

22-1 建立一个 CSV 文件

为了更详细地解说，笔者先用 ch22 文件夹的 report.xlsx 文件产生一个 CSV 文件，未来再用这个文件做说明。目前窗口内容是 report.xlsx，如下所示：

Name	Year	Product	Price	Quantity	Revenue	Location
Diana	2015	Black Tea	10	600	6000	New York
Diana	2015	Green Tea	7	660	4620	New York
Diana	2016	Black Tea	10	750	7500	New York
Diana	2016	Green Tea	7	900	6300	New York
Julia	2015	Black Tea	10	1200	12000	New York
Julia	2016	Black Tea	10	1260	12600	New York
Steve	2015	Black Tea	10	1170	11700	Chicago
Steve	2015	Green Tea	7	1260	8820	Chicago
Steve	2016	Black Tea	10	1350	13500	Chicago
Steve	2016	Green Tea	7	1440	10080	Chicago

执行“文件”→“另存为”命令，然后选择 D:\Python\ch22 文件夹。保存类型选 CSV（逗号分隔）(*.csv)，然后将文件名改为 csvReport。按“保存”按钮后，会弹出对话框确认是否要继续使用该格式，选择“是”，可以得到下列结果。

Name	Year	Product	Price	Quantity	Revenue	Location
Diana	2015	Black Tea	10	600	6000	New York
Diana	2015	Green Tea	7	660	4620	New York
Diana	2016	Black Tea	10	750	7500	New York
Diana	2016	Green Tea	7	900	6300	New York
Julia	2015	Black Tea	10	1200	12000	New York
Julia	2016	Black Tea	10	1260	12600	New York
Steve	2015	Black Tea	10	1170	11700	Chicago
Steve	2015	Green Tea	7	1260	8820	Chicago
Steve	2016	Black Tea	10	1350	13500	Chicago
Steve	2016	Green Tea	7	1440	10080	Chicago

可见一个 CSV 文件已经成功建立了，文件名是 csvReport.csv，可以关闭上述 Excel 窗口了。

22-2　用记事本打开 CSV 文件

CSV 文件的特色是几乎可以在所有不同的电子表格内编辑，当然也可以在一般的文字编辑程序内查阅使用。如果我们现在使用记事本打开这个 CSV 文件，可以看到这个文件的原貌。

```
Name,Year,Product,Price,Quantity,Revenue,Location
Diana,2015,Black Tea,10,600,6000,New York
Diana,2015,Green Tea,7,660,4620,New York
Diana,2016,Black Tea,10,750,7500,New York
Diana,2016,Green Tea,7,900,6300,New York
Julia,2015,Black Tea,10,1200,12000,New York
Julia,2016,Black Tea,10,1260,12600,New York
Steve,2015,Black Tea,10,1170,11700,Chicago
Steve,2015,Green Tea,7,1260,8820,Chicago
Steve,2016,Black Tea,10,1350,13500,Chicago
Steve,2016,Green Tea,7,1440,10080,Chicago
```

22-3　CSV 模块

Python 有内建 CSV 模块，导入这个模块后，可以很轻松地读取 CSV 文件，方便未来程序操作，所以本章程序前端要加上下列指令：

```
import csv
```

22-4　读取 CSV 文件

22-4-1　使用 open( ) 打开 CSV 文件

在读取 CSV 文件前，第一步是使用 open( ) 打开文件，语法格式如下：

```
with open(文件名) as csvFile # csvFile 是可以自行命名的文件对象相关系列指令
```

如果忘了 with 关键词的用法，可以参考 14-2-2 节。当然你也可以直接使用传统方法打开文件。

```
csvFile = open(文件名)          # 打开文件建立 CSV 文件对象 csvFile
```

22-4-2　建立 Reader 对象

有了 CSV 文件对象后，下一步是可以使用 CSV 模块的 reader( ) 建立 Reader 对象，使用 Python 可以使用 list( ) 将这个 Reader 对象转换成列表（list），然后就可以很轻松地使用这个列表资料了。

程序实例 ch22_1.py：打开 csvReport.csv 文件，读取 CSV 文件建立 Reader 对象 csvReader，再将 csvReader 对象转成列表数据，然后打印列表数据。

```
1  # ch22_1.py
2  import csv
3
4  fn = 'csvReport.csv'
5  with open(fn) as csvFile:                # 打开csv文件
6      csvReader = csv.reader(csvFile)      # 读取文件建立Reader对象
7      listReport = list(csvReader)         # 将数据转成列表
8  print(listReport)                        # 打印列表数据
```

执行结果

```
===================== RESTART: D:\Python\ch22\ch22_1.py =====================
[['Name', 'Year', 'Product', 'Price', 'Quantity', 'Revenue', 'Location'], ['Dian
a', '2015', 'Black Tea', '10', '600', '6000', 'New York'], ['Diana', '2015', 'Gr
een Tea', '7', '660', '4620', 'New York'], ['Diana', '2016', 'Black Tea', '10',
'750', '7500', 'New York'], ['Diana', '2016', 'Green Tea', '7', '900', '6300', '
New York'], ['Julia', '2015', 'Black Tea', '10', '1200', '12000', 'New York'], [
'Julia', '2016', 'Black Tea', '10', '1260', '12600', 'New York'], ['Steve', '201
5', 'Black Tea', '10', '1170', '11700', 'Chicago'], ['Steve', '2015', 'Green Tea
', '7', '1260', '8820', 'Chicago'], ['Steve', '2016', 'Black Tea', '10', '1350',
 '13500', 'Chicago'], ['Steve', '2016', 'Green Tea', '7', '1440', '10080', 'Chic
ago']]
```

上述程序需留意的是，程序第 6 行所建立的 Reader 对象 csvReader，只能在 with 关键区块内使用，此例是 5 ～ 7 行，未来我们要继续操作这个 CSV 文件内容，需使用第 7 行所建的列表 listReport 或是重新打开文件与读取文件。

22-4-3 用循环列出 Reader 对象数据

我们可以使用 for 循环操作 Reader 对象，列出各行数据，同时使用 Reader 对象的 line_num 属性列出行号。

程序实例 ch22_2.py：读取 Reader 对象，然后以循环方式列出对象内容。

```
1  # ch22_2.py
2  import csv
3
4  fn = 'csvReport.csv'
5  with open(fn) as csvFile:                # 打开csv文件
6      csvReader = csv.reader(csvFile)      # 读取文件建立Reader对象csvReader
7      for row in csvReader:                # 用循环列出csvReader对象内容
8          print("Row %s = " % csvReader.line_num, row)
```

执行结果

```
===================== RESTART: D:\Python\ch22\ch22_2.py =====================
Row 1 =  ['Name', 'Year', 'Product', 'Price', 'Quantity', 'Revenue', 'Location']
Row 2 =  ['Diana', '2015', 'Black Tea', '10', '600', '6000', 'New York']
Row 3 =  ['Diana', '2015', 'Green Tea', '7', '660', '4620', 'New York']
Row 4 =  ['Diana', '2016', 'Black Tea', '10', '750', '7500', 'New York']
Row 5 =  ['Diana', '2016', 'Green Tea', '7', '900', '6300', 'New York']
Row 6 =  ['Julia', '2015', 'Black Tea', '10', '1200', '12000', 'New York']
Row 7 =  ['Julia', '2016', 'Black Tea', '10', '1260', '12600', 'New York']
Row 8 =  ['Steve', '2015', 'Black Tea', '10', '1170', '11700', 'Chicago']
Row 9 =  ['Steve', '2015', 'Green Tea', '7', '1260', '8820', 'Chicago']
Row 10 =  ['Steve', '2016', 'Black Tea', '10', '1350', '13500', 'Chicago']
Row 11 =  ['Steve', '2016', 'Green Tea', '7', '1440', '10080', 'Chicago']
```

22-4-4 用循环列出列表内容

for 循环也可用于列出列表内容。

程序实例 ch22_3.py：用 for 循环列出列表内容。

```
1  # ch22_3.py
2  import csv
3
4  fn = 'csvReport.csv'
5  with open(fn) as csvFile:                # 打开csv文件
6      csvReader = csv.reader(csvFile)     # 读取文件建立Reader对象
7      listReport = list(csvReader)        # 将数据转成列表
8  for row in listReport:                  # 使用循环列出列表内容
9      print(row)
```

执行结果

```
===================== RESTART: D:\Python\ch22\ch22_3.py =====================
['Name', 'Year', 'Product', 'Price', 'Quantity', 'Revenue', 'Location']
['Diana', '2015', 'Black Tea', '10', '600', '6000', 'New York']
['Diana', '2015', 'Green Tea', '7', '660', '4620', 'New York']
['Diana', '2016', 'Black Tea', '10', '750', '7500', 'New York']
['Diana', '2016', 'Green Tea', '7', '900', '6300', 'New York']
['Julia', '2015', 'Black Tea', '10', '1200', '12000', 'New York']
['Julia', '2016', 'Black Tea', '10', '1260', '12600', 'New York']
['Steve', '2015', 'Black Tea', '10', '1170', '11700', 'Chicago']
['Steve', '2015', 'Green Tea', '7', '1260', '8820', 'Chicago']
['Steve', '2016', 'Black Tea', '10', '1350', '13500', 'Chicago']
['Steve', '2016', 'Green Tea', '7', '1440', '10080', 'Chicago']
```

22-4-5　使用列表索引读取 CSV 内容

其实我们也可以使用第 6 章所学的列表知识，读取 CSV 内容。

程序实例 ch22_4.py：使用索引列出列表内容。

```
1  # ch22_4.py
2  import csv
3
4  fn = 'csvReport.csv'
5  with open(fn) as csvFile:               # 打开csv文件
6      csvReader = csv.reader(csvFile)    # 读取文件建立Reader对象
7      listReport = list(csvReader)       # 将数据转成列表
8
9  print(listReport[0][1], listReport[0][2])
10 print(listReport[1][2], listReport[1][5])
11 print(listReport[2][3], listReport[2][6])
```

执行结果

```
==================== RESTART: D:\Python\ch22\ch22_4.py ====================
Year Product
Black Tea 6000
7 New York
```

22-4-6　DictReader( )

这也是一个读取 CSV 文件的方法，不过返回的是排序字典（OrderedDict）类型，所以可以用域名当索引方式取得数据。在美国许多文件以 CSV 文件存储时，常常人名的 Last Name（姓）与 First Name（名）是分开以不同字段存储的，读取时可以使用这个方法，可参考 ch22 文件夹的 csvPeople.csv 文件。

```
first_name,last_name,city
Eli,Manning,New York
Kevin ,James,Cleveland
Mike,Jordon,Chicago
```

程序实例 ch22_5.py：使用 DictReader( ) 读取 CSV 文件，然后列出 DictReader 对象内容。

```
1  # ch22_5.py
2  import csv
3
4  fn = 'csvPeople.csv'
5  with open(fn) as csvFile:                       # 打开csv文件
6      csvDictReader = csv.DictReader(csvFile) # 读取文件建立DictReader对象
7      for row in csvDictReader:               # 列出DictReader各行内容
8          print(row)
```

执行结果

```
===================== RESTART: D:\Python\ch22\ch22_5.py =====================
OrderedDict([('first_name', 'Eli'), ('last_name', 'Manning'), ('city', 'New York
')])
OrderedDict([('first_name', 'Kevin '), ('last_name', 'James'), ('city', 'Clevela
nd')])
OrderedDict([('first_name', 'Mike'), ('last_name', 'Jordon'), ('city', 'Chicago'
)])
```

对于上述 OrderedDict 数据类型，可以使用下列方法读取。

程序实例 ch22_6.py：将 csvPeople.csv 文件的 last_name 与 first_name 解析出来。

```
1  # ch22_6.py
2  import csv
3
4  fn = 'csvPeople.csv'
5  with open(fn) as csvFile:                       # 打开csv文件
6      csvDictReader = csv.DictReader(csvFile) # 读取文件建立DictReader对象
7      for row in csvDictReader:               # 使用循环列出字典内容
8          print(row['first_name'], row['last_name'])
```

执行结果

```
===================== RESTART: D:\Python\ch22\ch22_6.py =====================
Eli Manning
Kevin  James
Mike Jordon
```

22-5 写入 CSV 文件

22-5-1 打开要写入的文件与关闭文件

想要将数据写入 CSV 文件，首先要打开一个文件供写入，如下所示：

```
csvFile = open('文件名', 'w', newline= '')        # w 是 write only 模式
...
csvFile.close( )                                  # 执行结束关闭文件
```

当然如果使用 with 关键词可以省略 close( )，如下所示：

```
with open('文件名', 'w', newline= '') as csvFile:
    ...
```

22-5-2　建立 writer 对象

如果应用前一节的 csvFile 对象，接下来需建立 writer 对象，语法如下：

```
with open('文件名', 'w', newline= '') as csvFile:
    outWriter = csv.writer(csvFile)
    …
```

或是

```
csvFile = open('文件名', 'w', newline= '')        # w 是 write only 模式
outWriter = csv.writer(csvFile)
…
csvFile.close( )                                  # 执行结束关闭文件
```

上述打开文件时多加参数 newline='', 可避免输出时每个行之间多空一行。

22-5-3　输出列表 writerow( )

writerow( ) 可以输出列表数据。

程序实例 ch22_7.py：输出列表数据的应用。

```
1  # ch22_7.py
2  import csv
3
4  fn = 'out22_7.csv'
5  with open(fn, 'w', newline = '') as csvFile:    # 打开csv文件
6      csvWriter = csv.writer(csvFile)              # 建立Writer对象
7      csvWriter.writerow(['Name', 'Age', 'City'])
8      csvWriter.writerow(['Hung', '35', 'Taipei'])
9      csvWriter.writerow(['James', '40', 'Chicago'])
```

执行结果

下列是分别用记事本与 Excel 打开文件的结果。

```
Name,Age,City
Hung,35,Taipei
James,40,Chicago
```

	A	B	C
1	Name	Age	City
2	Hung	35	Taipei
3	James	40	Chicago
4			

本书在 ch22 文件夹内有 ch22_7_1.py 文件，这个文件在第 5 行 open( ) 中没有加上 newline='', 造成输出时若用 Excel 窗口观察有跳行输出的结果，可参考 out22_7_1.csv 文件。至于用记事本打开文件则一切正常，下列是程序代码。

```
5  with open(fn, 'w') as csvFile:                  # 打开csv文件
```

下列是执行结果，读者可以比较下图右边的 Excel 表。

```
Name,Age,City
Hung,35,Taipei
James,40,Chicago
```

	A	B	C
1	Name	Age	City
2			
3	Hung	35	Taipei
4			
5	James	40	Chicago

程序实例 ch22_8.py：复制 CSV 文件，这个程序会读取文件，然后将文件写入另一个文件方式，达成复制的目的。

```
1  # ch22_8.py
2  import csv
3
4  infn = 'csvReport.csv'                                   # 来源文件
5  outfn = 'out22_8.csv'                                    # 目标文件
6  with open(infn) as csvRFile:                             # 打开csv文件供读取
7      csvReader = csv.reader(csvRFile)                     # 读取文件建立Reader对象
8      listReport = list(csvReader)                         # 将数据转成列表
9
10 with open(outfn, 'w', newline = '') as csvOFile:         # 打开csv文件供写入
11     csvWriter = csv.writer(csvOFile)                     # 建立Writer对象
12     for row in listReport:                               # 将列表写入
13         csvWriter.writerow(row)
```

执行结果

读者可以打开 out22_8.csv 文件，内容将和 csvReport.csv 文件相同。

22-5-4　delimiter 关键词

delimiter 是分隔符，这个关键词用在 writer( ) 方法内。将数据写入 CSV 文件时，预设是同一行各栏间是逗号，可以用这个分隔符更改各栏间的逗号。

程序实例 ch22_9.py：将分隔符改为定位点字符（\t）。

```
1  # ch22_9.py
2  import csv
3
4  fn = 'out22_9.csv'
5  with open(fn, 'w', newline = '') as csvFile:                  # 打开csv文件
6      csvWriter = csv.writer(csvFile, delimiter='\t')           # 建立Writer对象
7      csvWriter.writerow(['Name', 'Age', 'City'])
8      csvWriter.writerow(['Hung', '35', 'Taipei'])
9      csvWriter.writerow(['James', '40', 'Chicago'])
```

执行结果

下列是用记事本打开 out22_9.csv 的结果。

```
Name    Age     City
Hung    35      Taipei
James   40      Chicago
```

当用 \t 字符取代逗号后，用 Excel 打开这个文件时，会将每行数据挤在一起，所以最好用记事本打开这类 CSV 文件。

22-5-5　写入字典数据 DictWriter( )

DictWriter( ) 可以写入字典数据，其语法格式如下：

```
dictWriter = csv.DictWriter(csvFile, fieldnames=fields)
```

上述 dictWriter 是字典的 Writer 对象，在进行上述指令前我们需要先设置 fields 列表，这个列表将包含未来字典内容的键（key）。

程序实例 ch22_10.py：使用 DictWriter( ) 将字典数据写入 CSV 文件。

```
1  # ch22_10.py
2  import csv
3
4  fn = 'out22_10.csv'
5  with open(fn, 'w', newline = '') as csvFile:                # 打开csv文件
6      fields = ['Name', 'Age', 'City']
7      dictWriter = csv.DictWriter(csvFile, fieldnames=fields)  # 建立Writer对象
8
9      dictWriter.writeheader()                                 # 写入标题
10     dictWriter.writerow({'Name':'Hung', 'Age':'35', 'City':'Taipei'})
11     dictWriter.writerow({'Name':'James', 'Age':'40', 'City':'Chicago'})
```

执行结果

下列是用 Excel 打开 out22_10.csv 的结果。

	A	B	C	D
1	Name	Age	City	
2	Hung	35	Taipei	
3	James	40	Chicago	
4				

上述程序第 9 行的 writeheader( ) 主要是写入我们在第 7 行设置的 fieldname。

程序实例 ch22_11.py：改写程序实例 ch22_10.py，将要写入 CSV 文件的数据改成列表数据，此列表数据的元素是字典。

```
1  # ch22_11.py
2  import csv
3
4  dictList = [{'Name':'Hung', 'Age':'35', 'City':'Taipei'},      # 定义列表,元素是字典
5              {'Name':'James', 'Age':'40', 'City':'Chicago'}]
6
7  fn = 'out22_11.csv'
8  with open(fn, 'w', newline = '') as csvFile:                   # 打开csv文件
9      fields = ['Name', 'Age', 'City']
10     dictWriter = csv.DictWriter(csvFile, fieldnames=fields)    # 建立Writer对象
11
12     dictWriter.writeheader()                                   # 写入标题
13     for row in dictList:                                       # 写入内容
14         dictWriter.writerow(row)
```

执行结果

打开 out22_11.csv 后与 out22_10.csv 相同。

22-6　专题——使用 CSV 文件绘制气象图表

其实网络上有许多 CSV 文件，原始的文件有些复杂，不过我们可以使用 Python 读取文件，然后筛选需要的字段，整个工作就变得比较简单了。本节主要是用实例介绍将图表设计应用在 CSV 文件中。

22-6-1　台北市 2017 年 1 月气象资料

在 ch22 文件夹内有 TaipeiWeatherJan.csv 文件，这是记录了 2017 年 1 月份台北市的气象资料，这个文件的 Excel 内容如下：

	A	B	C	D	E	F	G
1	Date	HighTemperature	MeanTemperature	LowTemperature			
2	2017/1/1	26	23	20			
3	2017/1/2	25	22	18			
4	2017/1/3	22	20	19			
5	2017/1/4	27	24	20			
6	2017/1/5	25	22	19			
7	2017/1/6	25	22	20			
8	2017/1/7	26	23	20			
9	2017/1/8	22	20	18			
10	2017/1/9	18	16	17			
11	2017/1/10	20	18	16			

程序实例 ch22_12.py：读取 TaipeiWeatherJan.csv 文件，然后列出标题栏。

```
1  # ch22_12.py
2  import csv
3
4  fn = 'TaipeiWeatherJan.csv'
5  with open(fn) as csvFile:
6      csvReader = csv.reader(csvFile)
7      headerRow = next(csvReader)          # 读取文件下一行
8  print(headerRow)
```

执行结果

```
===================== RESTART: D:\Python\ch22\ch22_12.py =====================
['Date', 'HighTemperature', 'MeanTemperature', 'LowTemperature']
```

从上图我们可以得到 TaipeiWeatherJan.csv 有 4 个字段，分别是记载日期（Date）、当天最高温（HighTemperature）、平均温度（MeanTemperature）、最低温度（LowTemperature）。上述第 7 行的 next( ) 可以读取下一行。

22-6-2　列出标题数据

我们可以使用 6-12 节所介绍的 enumerate( )。

程序实例 ch22_13.py：列出 TaipeiWeatherJan.csv 文件的标题与相对应的索引。

```
1  # ch22_13.py
2  import csv
3
4  fn = 'TaipeiWeatherJan.csv'
5  with open(fn) as csvFile:
6      csvReader = csv.reader(csvFile)
7      headerRow = next(csvReader)          # 读取文件下一行
8  for i, header in enumerate(headerRow):
9      print(i, header)
```

执行结果

```
===================== RESTART: D:\Python\ch22\ch22_13.py =====================
0 Date
1 HighTemperature
2 MeanTemperature
3 LowTemperature
```

22-6-3　读取最高温与最低温

程序实例 ch22_14.py：读取 TaipeiWeatherJan.csv 文件的最高温与最低温。这个程序会将一月份的最高温放在 highTemps 列表，最低温放在 lowTemps 列表。

```
 1  # ch22_14.py
 2  import csv
 3
 4  fn = 'TaipeiWeatherJan.csv'
 5  with open(fn) as csvFile:
 6      csvReader = csv.reader(csvFile)
 7      headerRow = next(csvReader)          # 读取文件下一行
 8      highTemps, lowTemps = [], []         # 设置空列表
 9      for row in csvReader:
10          highTemps.append(row[1])         # 存储最高温
11          lowTemps.append(row[3])          # 存储最低温
12
13  print("最高温 : ", highTemps)
14  print("最低温 : ", lowTemps)
```

执行结果

```
====================== RESTART: D:\Python\ch22\ch22_14.py ======================
最高温 :  ['26', '25', '22', '27', '25', '25', '26', '22', '18', '20', '21', '22
', '18', '15', '15', '16', '23', '23', '22', '18', '15', '17', '16', '17', '18',
 '19', '24', '26', '25', '27', '18']
最低温 :  ['20', '18', '19', '20', '19', '20', '20', '18', '17', '16', '18', '18
', '14', '12', '13', '13', '16', '18', '18', '12', '12', '12', '13', '14', '13',
 '13', '13', '16', '17', '14', '14']
```

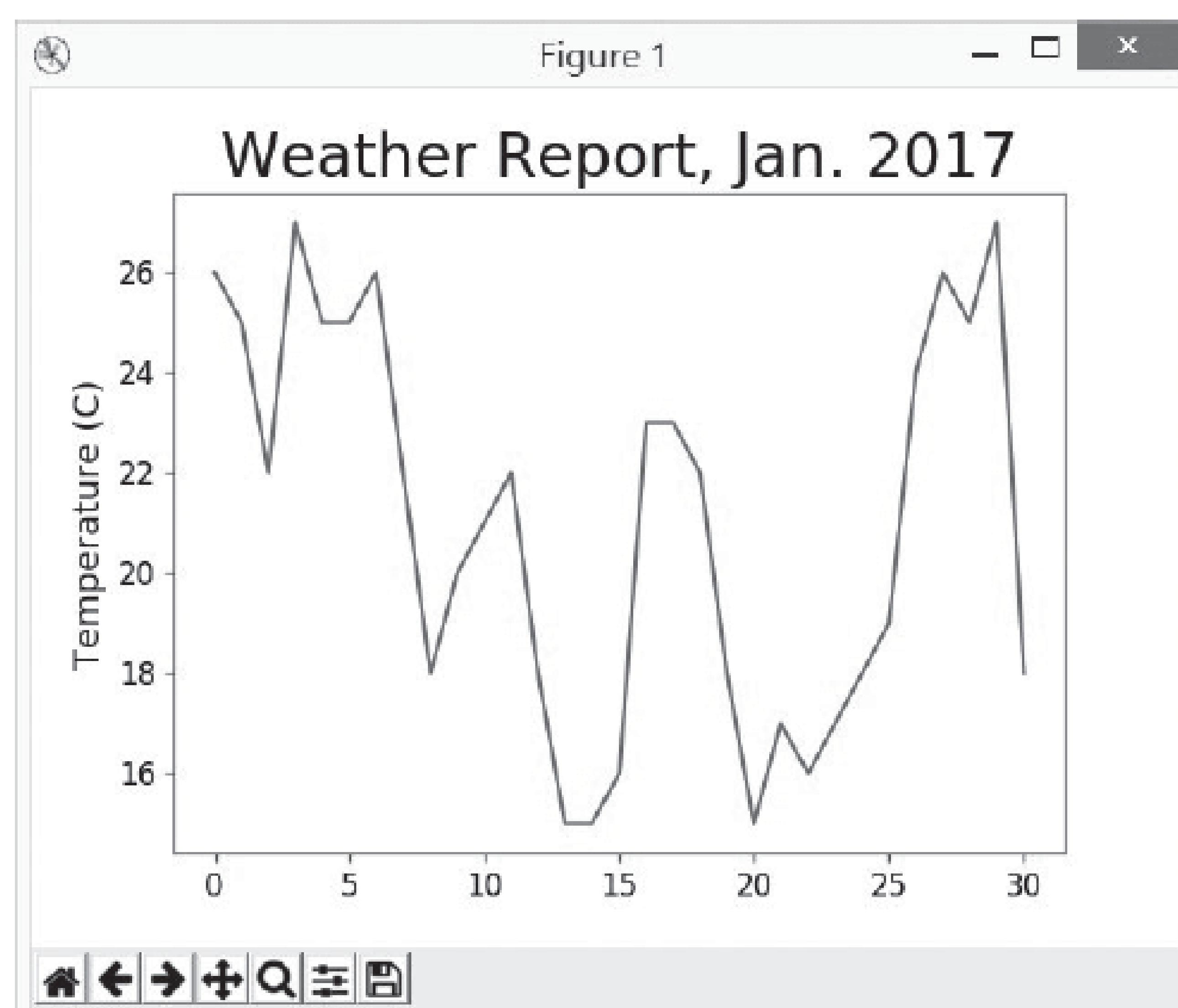
22-6-4　绘制最高温

其实这一节内容不复杂，所有绘图方法前面各小节已有说明。

程序实例 ch22_15.py：绘制 2017 年 1 月份台北市每天气温的最高温，请注意第 11 行存储温度时使用 int(row[1])，相当于用整数存储。

```
 1  # ch22_15.py
 2  import csv
 3  import matplotlib.pyplot as plt
 4
 5  fn = 'TaipeiWeatherJan.csv'
 6  with open(fn) as csvFile:
 7      csvReader = csv.reader(csvFile)
 8      headerRow = next(csvReader)          # 读取文件下一行
 9      highTemps = []                       # 设置空列表
10      for row in csvReader:
11          highTemps.append(int(row[1]))    # 存储最高温
12
13  plt.plot(highTemps)
14  plt.title("Weather Report, Jan. 2017", fontsize=24)
15  plt.xlabel("", fontsize=14)
16  plt.ylabel("Temperature (C)", fontsize=14)
17  plt.tick_params(axis='both', labelsize=12, color='red')
18  plt.show()
```

执行结果



22-6-5　设置绘图区大小

目前绘图区大小是使用系统默认，不过我们可以使用 figure( ) 设置绘图区大小，设置方式如下：

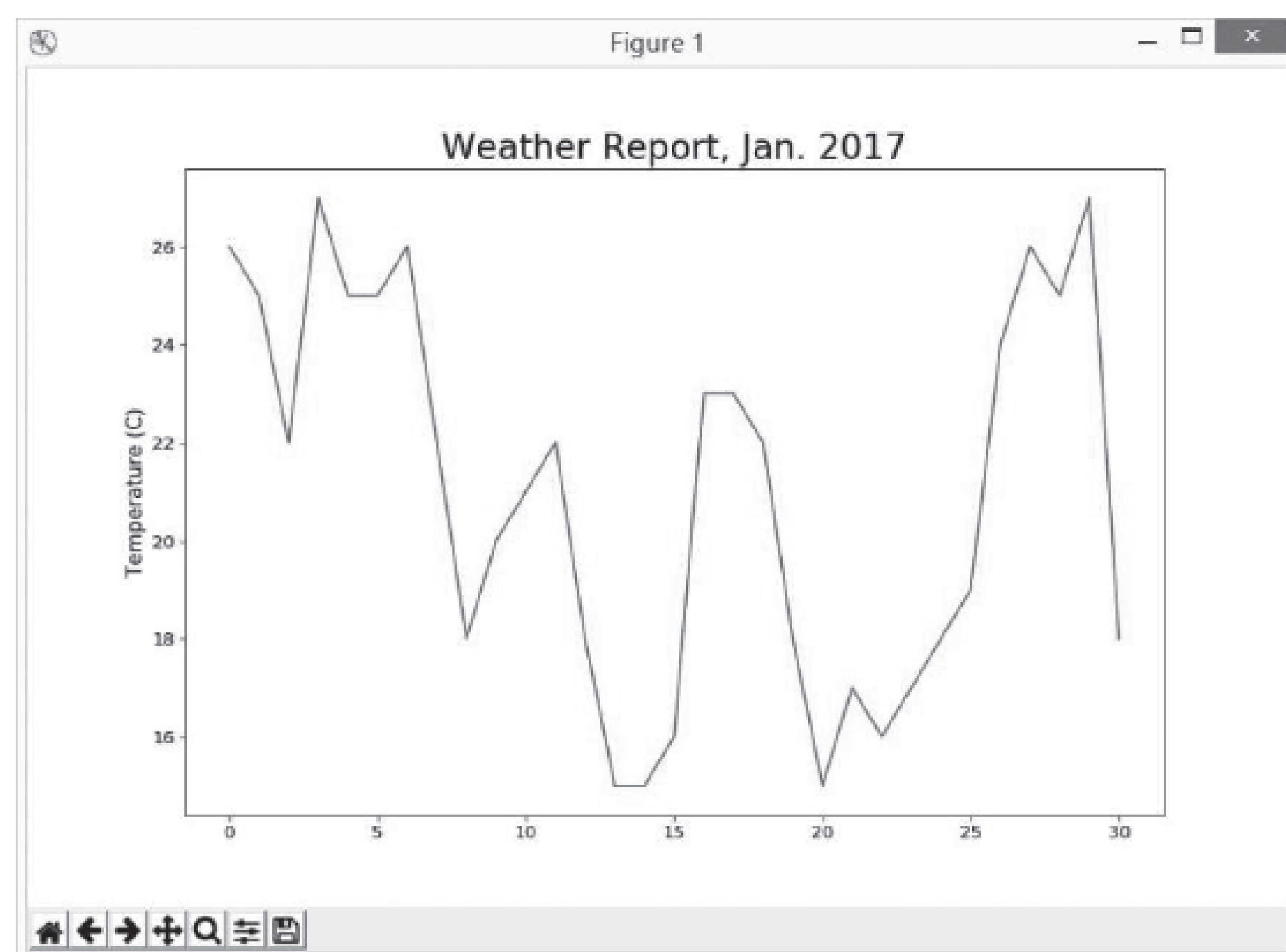
```
figure(dpi=n, figsize=(width, height))
```

经上述设置后，绘图区的宽将是 n*width 像素，高是 n*width 像素。

程序实例 ch22_16.py：重新设计 ch22_15.py，设置绘图区宽度是 960，高度是 640，这个程序只是增加下列行：

```
12  plt.figure(dpi=80, figsize=(12, 8))          # 设置绘图区大小
```

执行结果



22-6-6　日期格式

天气图表建立时，我们可能想把日期加在 x 轴的刻度上，这时我们需要使用 Python 内建的 datetime 模块，在使用前请使用下列方式导入模块：

```
from datetime import datetime
```

然后可以使用下列方法将日期字符串解析为日期对象：

strptime(string, format)

string 是要解析的日期字符串，format 是该日期字符串的目前格式，下表是日期格式参数的意义。

参数	说明
%Y	4 位数年份，例如 2017
%y	2 位数年份，例如 17
%m	月份（1 ～ 12）
%B	月份名称，例如 January
%A	星期名称，例如 Sunday
%d	日期（1 ～ 31）
%H	24 小时 (0 ～ 23)
%I	12 小时（1 ～ 12）
%p	AM 或 PM
%M	分钟（0 ～ 59）
%S	秒（0 ～ 59）

程序实例 ch22_17.py：将字符串转成日期对象。

```
1  # ch22_17.py
2  from datetime import datetime
3
4  dateObj = datetime.strptime('2017/1/1', '%Y/%m/%d')
5  print(dateObj)
```

执行结果

```
====================== RESTART: D:\Python\ch22\ch22_17.py ======================
2017-01-01 00:00:00
```

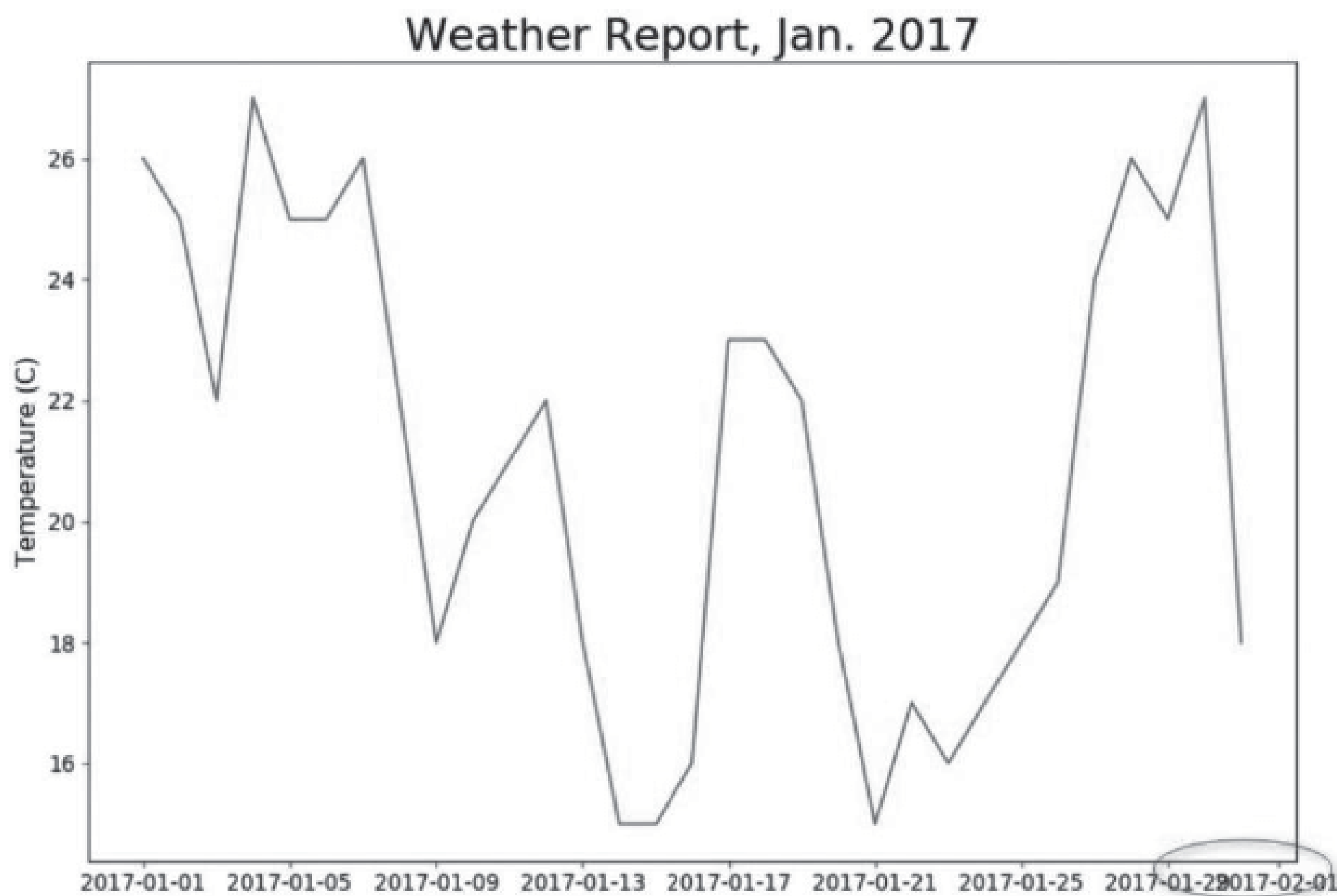
22-6-7　在图表增加日期刻度

其实在 plot( ) 方法内增加日期列表参数时，就可以在图表增加日期刻度。

程序实例 ch22_18.py：为图表增加日期刻度。

```
1  # ch22_18.py
2  import csv
3  import matplotlib.pyplot as plt
4  from datetime import datetime
5
6  fn = 'TaipeiWeatherJan.csv'
7  with open(fn) as csvFile:
8      csvReader = csv.reader(csvFile)
9      headerRow = next(csvReader)              # 读取文件下一行
10     dates, highTemps = [], []                # 设置空列表
11     for row in csvReader:
12         highTemps.append(int(row[1]))        # 存储最高温
13         currentDate = datetime.strptime(row[0], "%Y/%m/%d")
14         dates.append(currentDate)
15
16 plt.figure(dpi=80, figsize=(12, 8))          # 设置绘图区大小
17 plt.plot(dates, highTemps)                   # 图标增加日期刻度
18 plt.title("Weather Report, Jan. 2017", fontsize=24)
19 plt.xlabel("", fontsize=14)
20 plt.ylabel("Temperature (C)", fontsize=14)
21 plt.tick_params(axis='both', labelsize=12, color='red')
22 plt.show()
```

执行结果



这个程序的第一个重点是第 13 行和 14 行，主要是将日期字符串转成对象，然后存入 dates 日期列表。第二个重点是第 17 行，在 plot( ) 方法中第一个参数放 dates 日期列表。上述缺点是日期有重叠，可以参考下一节将日期旋转改良。

22-6-8 日期位置的旋转

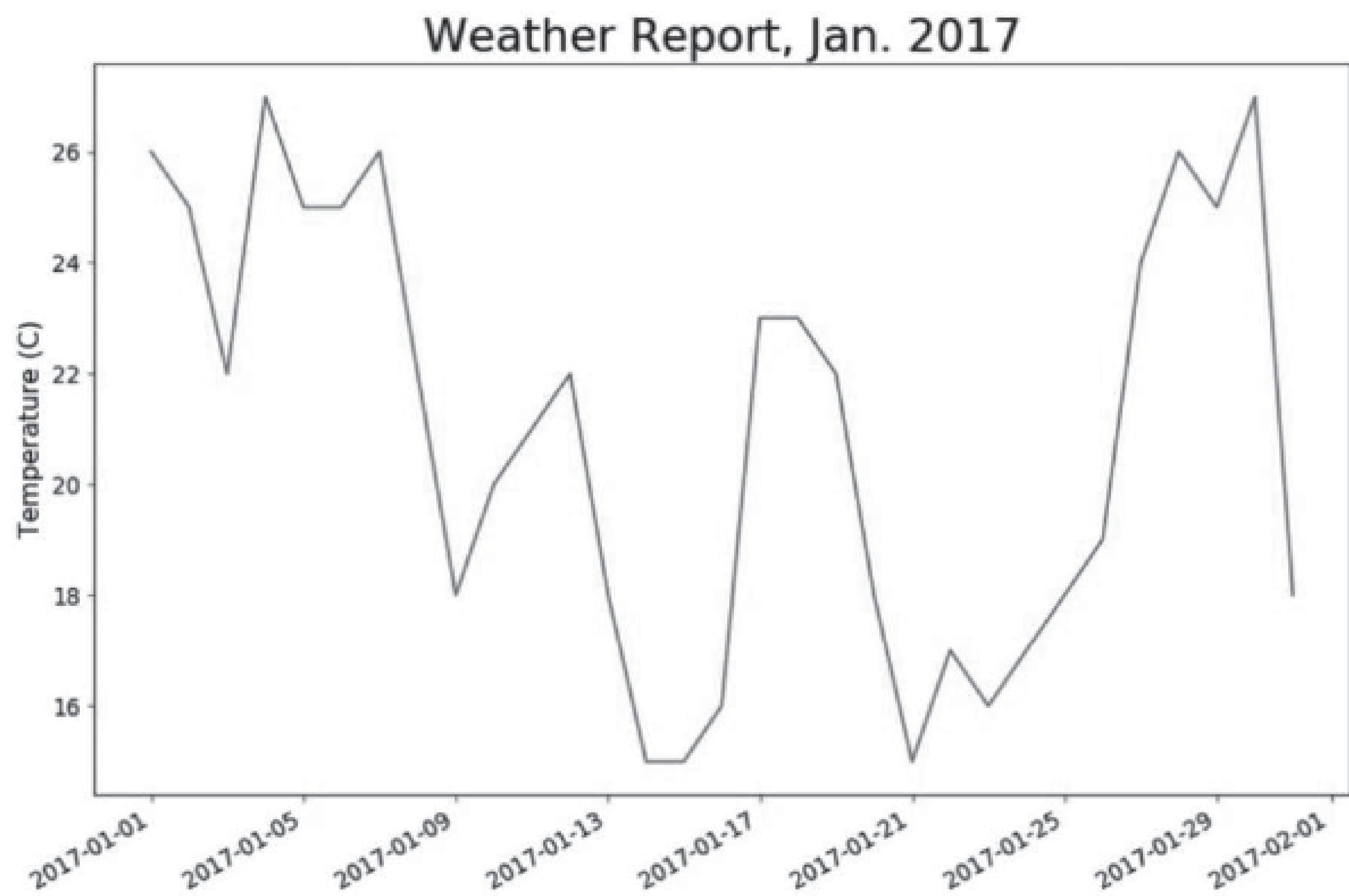
上一节的执行结果中可以发现日期是水平放置，可以用 autofmt_xdate( ) 设置日期旋转，语法如下：

```
fig = plt.figure( xxx )                # xxx 是相关设置信息
…
fig.autofmt_xdate(rotation=xx)         # rotation 若省略则系统使用优化默认
```

程序实例 ch22_19.py：重新设计 ch22_18.py，将日期旋转。

```
16  fig = plt.figure(dpi=80, figsize=(12, 8))   # 设置绘图区大小
17  plt.plot(dates, highTemps)                  # 图标增加日期刻度
18  fig.autofmt_xdate()                         # 日期旋转
```

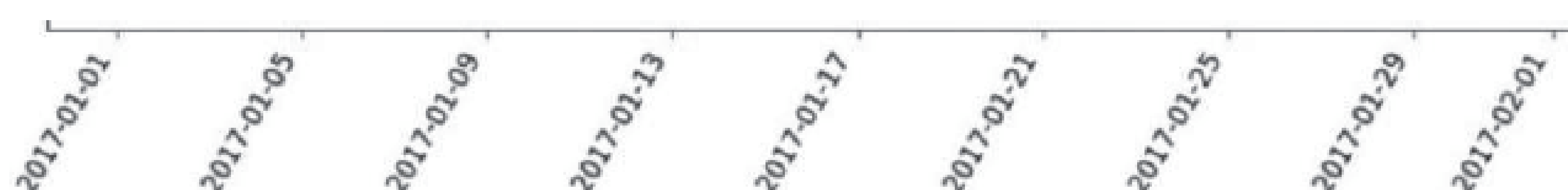
执行结果



程序实例 ch22_20.py：将日期字符串调整为旋转 60 度，只需增加下列行：

```
18  fig.autofmt_xdate(rotation=60)                    # 日期旋转
```

执行结果



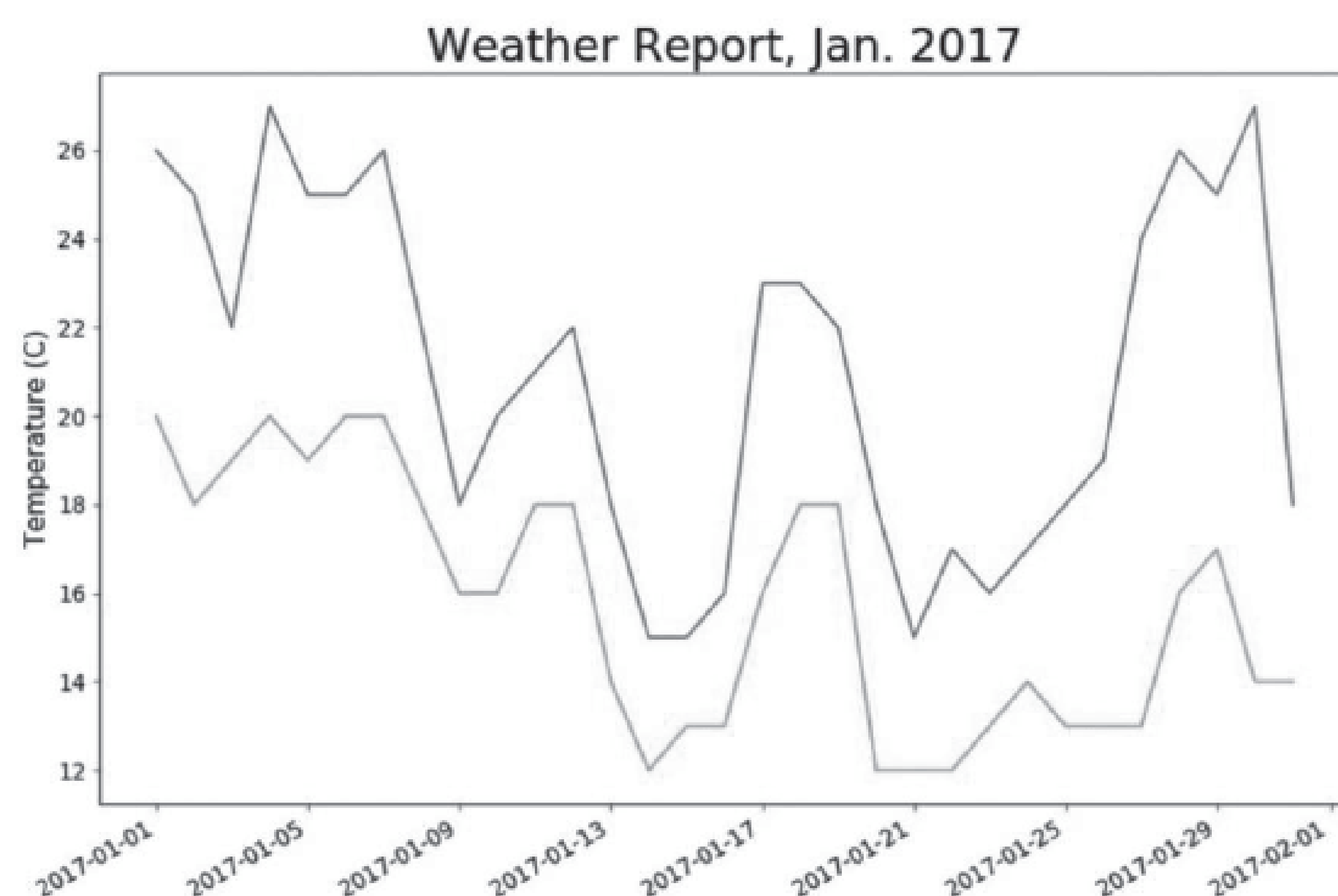
22-6-9　绘制最高温与最低温

在 TaipeiWeatherJan.csv 文件内有最高温与最低温的字段，下面将同时绘制最高温与最低温。

程序实例 ch22_21.py：绘制最高温与最低温。这个程序的第一个重点是程序第 11 ～ 21 行使用异常处理方式，因为读者在读取真实的网络数据时，常常会遇到不可预期的数据，例如数据少了或是数据格式错误，往往造成程序中断。为了避免程序因数据不良出错，可以使用异常处理方式。第二个重点是程序的第 24 行和第 25 行分别是绘制最高温与最低温。

```
1   # ch22_21.py
2   import csv
3   import matplotlib.pyplot as plt
4   from datetime import datetime
5
6   fn = 'TaipeiWeatherJan.csv'
7   with open(fn) as csvFile:
8       csvReader = csv.reader(csvFile)
9       headerRow = next(csvReader)                 # 读取文件下一行
10      dates, highTemps, lowTemps = [], [], []  # 设置空列表
11      for row in csvReader:
12          try:
13              currentDate = datetime.strptime(row[0], "%Y/%m/%d")
14              highTemp = int(row[1])              # 设置最高温
15              lowTemp = int(row[3])               # 设置最低温
16          except Exception:
17              print('有缺值')
18          else:
19              highTemps.append(highTemp)          # 存储最高温
20              lowTemps.append(lowTemp)            # 存储最低温
21              dates.append(currentDate)           # 存储日期
22
23  fig = plt.figure(dpi=80, figsize=(12, 8))    # 设置绘图区大小
24  plt.plot(dates, highTemps)                    # 绘制最高温
25  plt.plot(dates, lowTemps)                     # 绘制最低温
26  fig.autofmt_xdate()                           # 日期旋转
27  plt.title("Weather Report, Jan. 2017", fontsize=24)
28  plt.xlabel("", fontsize=14)
29  plt.ylabel("Temperature (C)", fontsize=14)
30  plt.tick_params(axis='both', labelsize=12, color='red')
31  plt.show()
```

执行结果



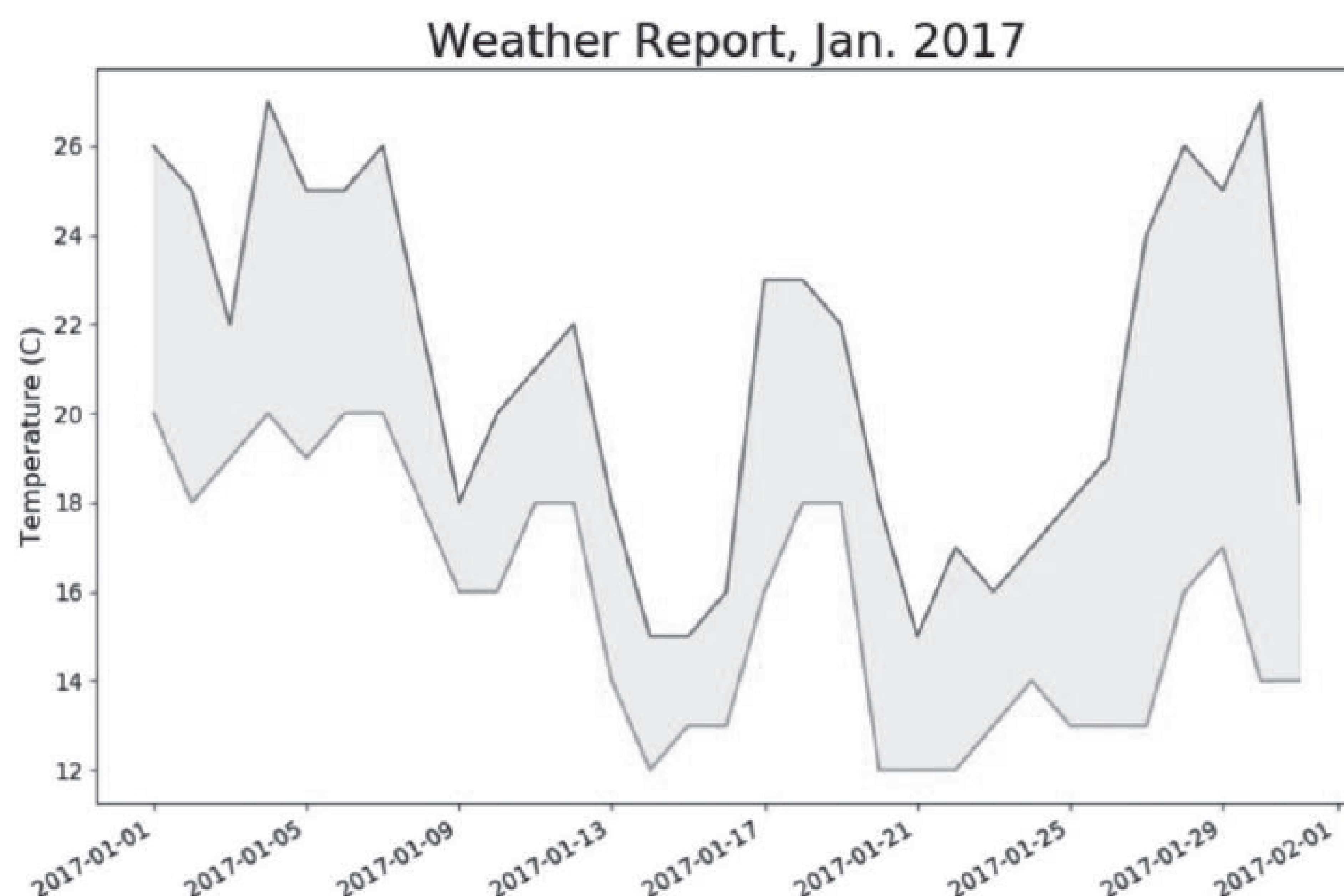
22-6-10　填满最高温与最低温之间的区域

可以使用 fill_between( ) 方法填满最高温与最低温之间的区域。

程序实例 ch22_22.py：使用透明度是 0.2 的黄色填满区域，只需增加下列行：

```
26  plt.fill_between(dates, highTemps, lowTemps, color='y', alpha=0.2) # 填满区间
```

执行结果



22-6-11　后记

读者可能会想，学习打开个别 CSV 文件的用处在哪里？现在是大数据时代，所有搜集来的数据无法完整地用某一种格式呈现，CSV 是电子表格和数据库最常用的资料格式，我们可以先将所搜集的各式文件转成 CSV，然后就可以使用 Python 读取所有的 CSV 文件，再选取需要的数据做分析。此外，也可以将 CSV 文件当作不同数据库间的桥梁或数据库与电子表格间的桥梁。

习题

1. 请参考 ch22 文件夹的 csvReport.csv 文件，分别计算 2015 年和 2016 年的业绩。(**22-4 节**)

```
====================== RESTART: D:/Python/ex/ex22_1.py ======================
Total Revenue of 2015 =  43140
Total Revenue of 2016 =  49980
```

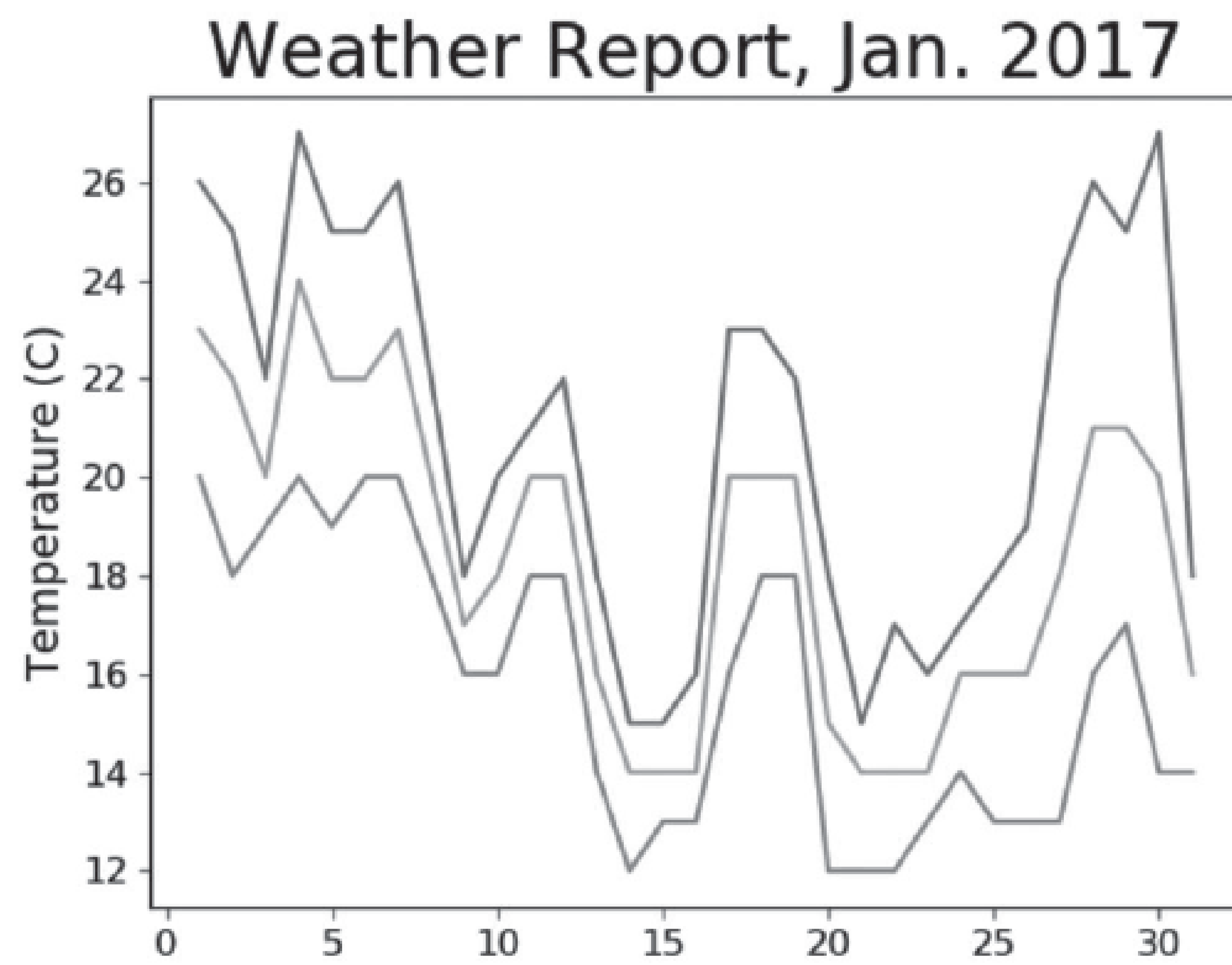
2. 请参考 ch22 文件夹的 csvReport.csv 文件，分别计算 Steve 在 2015 年和 2016 年的业绩。(**22-4 节**)

```
====================== RESTART: D:/Python/ex/ex22_2.py ======================
Steve's Total Revenue of 2015 =  20520
Steveis Total Revenue of 2016 =  23580
```

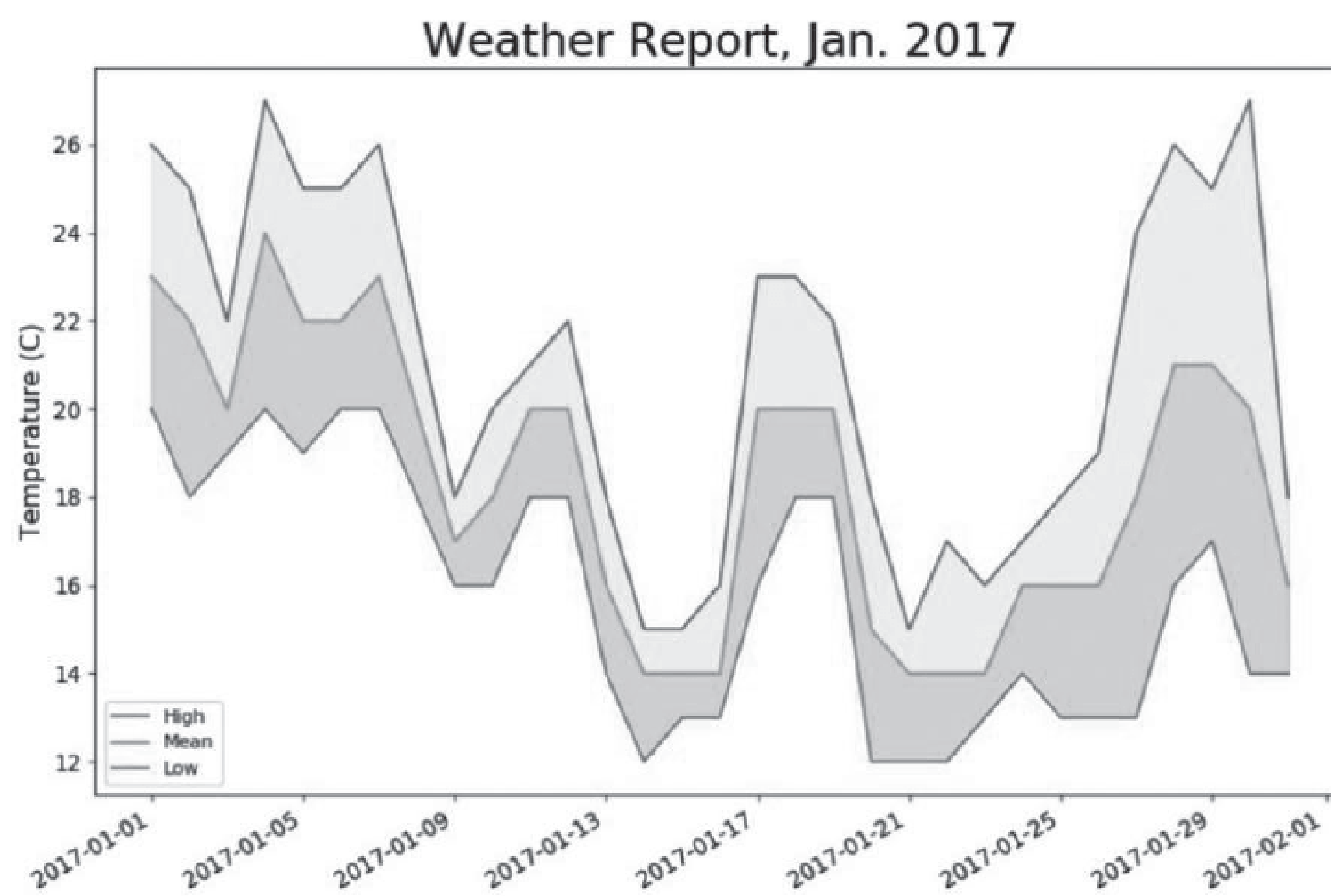
3. 请参考 ch22_14.py，增加列出平均温度。(**22-6 节**)

```
====================== RESTART: D:/Python/ex/ex22_3.py ======================
最高溫 :  ['26', '25', '22', '27', '25', '25', '26', '22', '18', '20', '21', '22
', '18', '15', '15', '16', '23', '23', '22', '18', '15', '17', '16', '17', '18',
 '19', '24', '26', '25', '27', '18']
平均溫 :  ['23', '22', '20', '24', '22', '22', '23', '20', '16', '18', '20', '20
', '16', '14', '14', '14', '20', '20', '20', '15', '14', '14', '14', '16', '16',
 '16', '18', '21', '21', '20', '16']
最低溫 :  ['20', '18', '19', '20', '19', '20', '20', '18', '17', '16', '18', '18
', '14', '12', '13', '13', '16', '18', '18', '12', '12', '12', '13', '14', '13',
 '13', '13', '16', '17', '14', '14']
```

4. 请参考 ch22_15.py，增加列出最高温和平均温。（**22-6 节**）



5. 请参考 ch22_22.py，但是需要增加图例，同时在最高温和平均温之间填满透明度为 0.2 的黄色，在平均温和最低温之间填满透明度为 0.2 的红色。（**22-6 节**）



23

第 2 3 章

Numpy 模块

本章摘要

Python 是一个应用范围很广的程序语言，第 6 章我们介绍了列表（list），第 8 章介绍了元组（tuple），我们可以使用它们执行一维数组（one-dimension array）或是多维数组（multi-dimension array）运算。虽然 list 或 tuple 弹性很大，很好用，但是如果使用高速计算时，伴随优点的同时也产生了一些缺点：

- 执行速度慢。
- 需要较多系统资源。

为此，许多追求高速运算的模块顺应诞生，这一章笔者将讲解在科学运算或人工智能领域最常见的、因高速运算而有的模块 Numpy，此名称所代表的是 **Num**erical **Py**thon。第 20 章与第 22 章中对此已经有些许说明，本章将做较完整的解说。

23-1 数组 ndarray

Numpy 模块所建立的数组数据形态称 ndarray（n-dimension array），n 代表维度，例如一维数组、二维数组、n 维数组。ndarray 数组的几个特色如下：

- 数组大小固定。
- 数组元素内容的数据形态相同。

也因为上述 Numpy 数组的特色，让它运算时可以有较好的执行速度，同时需要较少的系统资源。

23-2 Numpy 的数据形态

Numpy 支持比 Python 更多的数据形态，下列是 Numpy 所定义的数据形态：

- bool_：和 Python 的 bool 兼容，以一个字节储存 True 或 False。
- int_：默认的整数形态，与 C 语言的 long 相同，通常是 int32 或 int64。
- intc：与 C 语言的 int 相同，通常是 int32 或 int64。
- intp：用于索引的整数，与 C 语言的 size_t 相同，通常是 int32 或 int64。
- int8：8 位整数（-128-127）。
- int16：16 位整数（-32768-32767）。
- int32：32 位整数（-2147483648-2147483647）。
- int64：64 位整数（-9223372036854775808-9223372036854775807）。
- uint8：8 位无号整数（0-255）。
- uint16：16 位无号整数（0-65535）。
- uint32：32 位无号整数（0-4294967295）。
- uint64：64 位无号整数（0-18446744073709551615）。
- float_：与 Python 的 float 相同。
- float16：半精度浮点数，符号位，5 位指数，10 位尾数。
- float32：单精度浮点数，符号位，8 位指数，23 位尾数。

- ❑ float64：双倍精度浮点数，符号位，11 位指数，52 位尾数。
- ❑ complex_：复数，complex_128 的缩写。
- ❑ complex64：复数，由 2 个 32 位浮点数表示（实部和虚部）。
- ❑ complex128：复数，由 2 个 64 位浮点数表示（实部和虚部）。

23-3 一维数组

23-3-1 认识 ndarray 的属性

当使用 Numpy 模块建立 ndarray 数据形态的数组后，可以获得 ndarray 的属性，下列是几个常用的属性：

ndarray.dtype：数组元素形态。

ndarray.itemsize：数组元素数据形态大小（或称所占空间），单位是为字节。

ndarray.ndim：数组的维度。

ndarray.shape：数组维度元素个数的元组，也可以用于调整数组大小。

ndarray.size：数组元素个数。

23-3-2 建立一维数组

我们可以使用 array( ) 方法建立一维数组，建立时在小括号内填上中括号，然后将数组数值放在中括号内，彼此用逗号隔开。

实例 1：建立一维数组，数组内容是 1, 2, 3，同时列出数组的数据形态。

```
>>> import numpy as np
>>> x = np.array([1, 2, 3])
>>> print(type(x))          ← 打印x数据类型
<class 'numpy.ndarray'>
>>> print(x)                ← 打印x数组内容
[1 2 3]
```

数组建立好了，可以用索引方式取得或设置内容。

实例 2：列出数组元素内容。

```
>>> import numpy as np
>>> x = np.array([1, 2, 3])
>>> print(x[0])
1
>>> print(x[1])
2
>>> print(x[2])
3
```

实例 3：设置数组内容。

```
>>> import numpy as np
>>> x = np.array([1, 2, 3])
>>> x[1] = 10
>>> print(x)
[ 1 10  3]
```

实例 4：认识 ndarray 的属性。

```
>>> import numpy as np
>>> x = np.array([1, 2, 3])
>>> x.dtype              ←——————— 打印x数组元素形态
dtype('int32')
>>> x.itemsize           ←——————— 打印x数组元素大小
4
>>> x.ndim               ←——————— 打印x数组维度
1
>>> x.shape              ←——————— 打印x数组外形，3是第1维元素个数
(3,)
>>> x.size               ←——————— 打印x数组元素个数
3
```

上述 x.dtype 获得 int32，表示是 32 位的整数。x.itemsize 是数组元素大小，其中以字节为单位，一个字节是 8 位，由于元素是 32 位整数，所以返回是 4。x.ndim 返回数组维度是 1，表示这是一维数组。x.shape 以元组方式返回第一维元素个数是 3，未来二维数组还会解说。x.size 则是返回元素个数。

实例 5：array( ) 方法也可以接受使用 dtype 参数设置元素的数据形态。

```
>>> import numpy as np
>>> x = np.array([2, 4, 6], dtype=np.int8)
>>> x.dtype
dtype('int8')
```

实例 6：浮点数数组的建立与打印。

```
>>> import numpy as np
>>> y = np.array([1.1, 2.3, 3.6])
>>> y.dtype
dtype('float64')
>>> y
array([1.1, 2.3, 3.6])
>>> print(y)
[1.1 2.3 3.6]
```

其他常用建立一维数组的方法如下：

arange( )：建立相同等距的数组，可以参考 20-3-1 节。

linspace( )：可以参考 20-3-1 节。

下列是建立浮点数数组的方法：

zeros( )：默认是建立 0.0 浮点数的数组，不过可以使用 dtype 参数更改元素类型。

ones( )：默认是建立 1.0 浮点数的数组，不过可以使用 dtype 参数更改元素类型。

empty( )：默认是建立随机数浮点数的数组，不过可以使用 dtype 参数更改元素类型。

实例 7：使用 zeros( )，默认建立 5 个元素是 0.0 浮点数的数组。我们也可以使用 dtype 更改元素为整数。

```
>>> import numpy as np
>>> x = np.zeros(5)
>>> x.dtype
dtype('float64')
>>> print(x)
[0. 0. 0. 0. 0.]
>>> x = np.zeros(5, dtype=np.int_)
>>> x.dtype
dtype('int32')
>>> print(x)
[0 0 0 0 0]
```

实例 8：使用 ones( )，默认建立 5 个元素是 1.0 浮点数的数组。我们也可以使用 dtype 更改元素为整数。

```
>>> import numpy as np
>>> x = np.ones(5)
>>> x.dtype
dtype('float64')
>>> print(x)
[1. 1. 1. 1. 1.]
>>> x = np.ones(5, dtype=np.int32)
>>> x.dtype
dtype('int32')
>>> print(x)
[1 1 1 1 1]
```

23-3-3 一维数组的四则运算

我们可以将一般 Python 数学运算符号（+、-、*、/、//、%、**）应用在 Numpy 的数组。

实例 1：数组与整数的加法运算。

```
>>> import numpy as np
>>> x = np.array([1, 2, 3])
>>> y = x + 5
>>> print(y)
[6 7 8]
```

读者可以将上述概念应用在其他数学运算符号中。

实例 2：数组加法运算。

```
>>> import numpy as np
>>> x = np.array([1, 2, 3])
>>> y = np.array([10, 20, 30])
>>> z = x + y
>>> print(z)
[11 22 33]
```

实例 3：数组乘法运算。

```
>>> import numpy as np
>>> x = np.array([1, 2, 3])
>>> y = np.array([10, 20, 30])
>>> z = x * y
>>> print(z)
[10 40 90]
```

实例 4：数组除法运算。

```
>>> import numpy as np
>>> x = np.array([1, 2, 3])
>>> y = np.array([10, 20, 30])
>>> z = x / y
>>> print(z)
[0.1 0.1 0.1]
>>> z = y / x
>>> print(z)
[10. 10. 10.]
```

23-3-4 一维数组的关系运算符运算

在 5-1 节有关系运算符表，我们也可以将此运算符应用在数组运算。

实例：关系运算符应用在一维数组的运算。

```
>>> import numpy as np
>>> x = np.array([1, 2, 3])
>>> y = np.array([10, 20, 30])
>>> z = x > y
>>> print(z)
[False False False]
>>> z = x < y
>>> print(z)
[ True  True  True]
```

23-3-5　数组切片

在 6-1-3 节有介绍列表切片，那一节的切片概念也可以应用在数组。

实例：将切片应用在数组。

```
>>> import numpy as np
>>> x = np.array([1, 2, 3, 4, 5])
>>> print(x[0:3])
[1 2 3]
>>> print(x[1:4])
[2 3 4]
>>> print(x[0:5:2])
[1 3 5]
>>> print(x[-1])
5
>>> print(x[1:])
[2 3 4 5]
>>> print(x[:3])
[1 2 3]
```

23-3-6　数组结合或是加入数组元素

可以使用 concatenate( ) 将 2 个数组结合，或是将元素加入数组。

实例 1：将 2 个数组结合。

```
>>> import numpy as np
>>> x = np.array([1, 2, 3])
>>> y = np.array([4, 5])
>>> z = np.concatenate((x, y))
>>> print(z)
[1 2 3 4 5]
```

实例 2：将元素加入数组。

```
>>> import numpy as np
>>> x = np.array([1, 2, 3])
>>> z = np.concatenate((x, [4, 5]))
>>> print(z)
[1 2 3 4 5]
```

23-3-7　在数组指定索引位置插入元素

可以使用 insert（数组 , 索引 , 元素）在数组指定索引位置插入元素。

实例 1：在数组指定索引 2 插入元素 9。

```
>>> import numpy as np
>>> x = np.array([1, 2, 3, 4, 5])
>>> z = np.insert(x, 2, 9)
>>> print(z)
[1 2 9 3 4 5]
```

实例 2：在数组指定索引 1 和 3 分别插入元素 7 和 9。

```
>>> import numpy as np
>>> x = np.array([1, 2, 3, 4, 5])
>>> z = np.insert(x, [1, 3], [7, 9])
>>> print(z)
[1 7 2 3 9 4 5]
```

23-3-8　删除数组指定索引位置的元素

delete( ) 可以删除数组指定索引位置的元素。

实例 1：删除索引 1 的元素。

```
>>> import numpy as np
>>> x = np.array([1, 2, 3, 4, 5])
>>> z = np.delete(x, 1)
>>> print(z)
[1 3 4 5]
```

实例 2：删除索引 1 和 3 的元素。

```
>>> import numpy as np
>>> x = np.array([1, 2, 3, 4, 5])
>>> z = np.delete(x, [1, 3])
>>> print(z)
[1 3 5]
```

23-3-9　向量内积

有 2 个一维数组分别是 A（a1, a2, a3）和 B（b1, b2, b3），其向量内积（inner product）计算公式如下：

A • B = a1*b1 + a2*b2 + a3*b3

np.inner(A, B)　或 np.dot(A, B)

在人工智能的应用中，卷积（convolution）运算便是采用内积运算，其目的是取得图像特征，这是图像辨识的基础。

实例：计算一维数组的向量内积。

```
>>> import numpy as np
>>> x = np.array([1, 2, 3])
>>> y = np.array([4, 5, 6])
>>> z = np.inner(x, y)
>>> print(z)
32
>>> z = np.dot(x, y)
>>> print(z)
32
```

23-3-10　向量叉积

有 2 个一维数组分别是 A（a1, a2, a3）和 B（b1, b2, b3），其向量叉积（cross product）计算公式如下：

A x B =（a2*b3 – a3*b2, a3*b1 – a1*b3, a1*b2 - a2*b1）

np.cross（A, B）

实例：计算一维数组的向量叉积。

```
>>> import numpy as np
>>> x = np.array([1, 2, 3])
>>> y = np.array([4, 5, 6])
>>> z = np.cross(x, y)
>>> print(z)
[-3  6 -3]
```

23-3-11　向量外积

向量外积（outer product）的计算结果是一个矩阵，有 2 个一维数组分别是 A（a1, a2, a3）和 B（b1, b2, b3），其向量外积（outer product）计算公式如下：

$$A \cdot B = \begin{bmatrix} a1*b1 & a1*b2 & a1*b3 \\ a2*b1 & a2*b2 & a2*b3 \\ a3*b1 & a3*b2 & a3*b3 \end{bmatrix}$$

np.outer(A, B)

实例：计算一维数组的向量外积。

```
>>> import numpy as np
>>> x = np.array([1, 2, 3])
>>> y = np.array([4, 5, 6])
>>> z = np.outer(x, y)
>>> print(z)
[[ 4  5  6]
 [ 8 10 12]
 [12 15 18]]
```

23-3-12　将迭代运算应用在一维数组

程序实例 ch23_1.py：计算数组 [88, 92, 90, 0, 0] 的总和与平均，将总和填在索引 3，平均填在索引 4。

```
1  # ch23_1.py
2  import numpy as np
3
4  sum = 0
5  ave = 0
6  x = np.array([88, 92, 90, 0, 0])
7  for data in x:
8      sum += data
9  x[3] = sum
10 x[4] = sum / 3
11 print(x)
```

执行结果

```
==================== RESTART: D:/Python/ch23/ch23_1.py ====================
[ 88  92  90 270  90]
```

23-4　二维数组

在 6-7-3 节笔者有介绍二维列表，如下所示：

姓名	语文	英文	数学	总分
洪锦魁	80	95	88	0
洪冰儒	98	97	96	0
洪雨星	90	91	92	0
洪冰雨	91	93	95	0
洪星宇	92	97	90	0

上述分数部分可以处理成数组，由于数组只允许相同形态的数据存在，所以我们可以将分数部分处理成数组，此时存取数组方式如下：

姓名	语文	英文	数学	总分
洪锦魁	[0,0]	[0,1]	[0,2]	[0,3]
洪冰儒	[1,0]	[1,1]	[1,2]	[1,3]
洪雨星	[2,0]	[2,1]	[2,2]	[2,3]
洪冰雨	[3,0]	[3,1]	[3,2]	[3,3]
洪星宇	[4,0]	[4,1]	[4,2]	[4,3]

上述第 1 个索引是 row，第 2 个索引是 column，相当于是 [row, colomn]。

23-4-1　建立二维数组

建立二维数组与建立一维数组方法相同，可以使用 array( )，具体请参考下列实例。

实例 1：建立二维数组，同时列出数组的内容。

```
>>> import numpy as np
>>> x = np.array([[1, 2, 3],[4, 5, 6]])
>>> print(type(x))
<class 'numpy.ndarray'>
>>> print(x)
[[1 2 3]
 [4 5 6]]
```

实例 2：认识 ndarray 的属性。

```
>>> import numpy as np
>>> x = np.array([[1, 2, 3],[4, 5, 6]])
>>> x.dtype
dtype('int32')
>>> x.itemsize
4
>>> x.ndim
2
>>> x.shape
(2, 3)
>>> x.size
6
```

上述 x.ndim 返回 2，表示这是二维数组。x.shape 返回（2, 3），表示这是二维，每个维度有 3 个元素。

实例 3：与一维数组概念相同，array( ) 方法也可以接受使用 dtype 参数设置元素的数据形态。

```
>>> import numpy as np
>>> x = np.array([[1, 2, 3],[4, 5, 6]], dtype=np.int8)
>>> x.dtype
dtype('int8')
```

建立一维数组时所使用的 zeros( )、ones( )、empty( ) 方法也可以应用在二维数组。

实例 4：使用 zeros( ) 建立 2×3 数组。

```
>>> import numpy as np
>>> x = np.zeros((2, 3))
>>> print(x)
[[0. 0. 0.]
 [0. 0. 0.]]
```

实例 5：使用 ones( ) 建立 2×3 数组。

```
>>> import numpy as np
>>> x = np.ones((2, 3))
>>> print(x)
[[1. 1. 1.]
 [1. 1. 1.]]
```

23-4-2　二维数组相对位置的四则运算

二维数组四则运算的概念与一维数组相同。

实例 1：二维数组与整数的加法运算。

```
>>> import numpy as np
>>> x = np.array([[1, 2, 3],[4, 5, 6]])
>>> y = x + 10
>>> print(y)
[[11 12 13]
 [14 15 16]]
```

读者可以将上述概念应用在其他数学运算符号。

实例 2：二维数组加法运算。

```
>>> import numpy as np
>>> x = np.array([[1, 2],[3, 4]])
>>> y = np.array([[5, 6],[7, 8]])
>>> z = x + y
>>> print(z)
[[ 6  8]
 [10 12]]
```

实例 3：二维数组相对位置乘法运算。

```
>>> import numpy as np
>>> x = np.array([[1, 2],[3, 4]])
>>> y = np.array([[5, 6],[7, 8]])
>>> z = x * y
>>> print(z)
[[ 5 12]
 [21 32]]
```

需要留意，上述的“二维数组相对位置乘法”，与数学领域的矩阵乘法定义不一样，笔者将在 23-3-10 节说明矩阵乘法。

实例 4：二维数组除法运算。

```
>>> import numpy as np
>>> x = np.array([[10,20],[30,40]])
>>> y = np.array([[1,2],[3,4]])
>>> z = x / y
>>> print(z)
[[10. 10.]
 [10. 10.]]
```

23-4-3　二维数组的关系运算符运算

我们也可以将关系运算符应用在二维数组运算。

实例：关系运算符应用在二维数组的运算。

```
>>> import numpy as np
>>> x = np.array([[1, 2],[3, 4]])
>>> y = np.array([[5, 6],[7, 8]])
>>> z = x > y
>>> print(z)
[[False False]
 [False False]]
>>> z = x < y
>>> print(z)
[[ True  True]
 [ True  True]]
```

23-4-4　取得与设置二维数组元素

在 23-4 节笔者已经说明取得二维数组元素的方法，基本概念是用 [row,cloumn] 索引方式处理，下列是实例。

实例 1：取得二维数组某元素内容。

```
>>> import numpy as np
>>> x = np.array([[1, 2, 3],[4, 5, 6]])
>>> print(x[0,2])
3
>>> print(x[1,1])
5
```

实例 2：设置二维数组某元素内容。

```
>>> import numpy as np
>>> x = np.array([[1, 2, 3],[4, 5, 6]])
>>> x[1,2] = 10
>>> print(x)
[[ 1  2  3]
 [ 4  5 10]]
```

取得特定 row 的元素，例如，row=0 的元素可以写成 [0]、[0,]、[0,:]。

实例 3：取得特定 row=0 的元素。

```
>>> import numpy as np
>>> x = np.array([[1, 2, 3],[4, 5, 6]])
>>> print(x[0])
[1 2 3]
>>> print(x[0,])
[1 2 3]
>>> print(x[0,:])
[1 2 3]
```

取得特定 column 的元素，例如，column=0 的元素可以写成 [:,0]。

实例 4：取得特定 column=0 的元素。

```
>>> import numpy as np
>>> x = np.array([[1, 2, 3],[4, 5, 6]])
>>> print(x[:,0])
[1 4]
```

23-4-5 二维数组切片

切片的概念可以应用在二维数组。

实例 1：将切片应用在二维数组，取得 row=0 的前 3 个元素。

```
>>> import numpy as np
>>> x = np.array([[1, 2, 3, 4],[2, 3, 4, 5], [3, 4, 5, 6]])
>>> print(x[0:3,0])
[1 2 3]
```

实例 2：将切片应用在二维数组，取得 row=0:2，column=2:4 的元素。

```
>>> import numpy as np
>>> x = np.array([[1, 2, 3, 4],[2, 3, 4, 5],[3, 4, 5, 6]])
>>> print(x[0:2,2:4])
[[3 4]
 [4 5]]
```

实例 3：取得前 2 个 row 的元素。

```
>>> import numpy as np
>>> x = np.array([[1, 2, 3, 4],[2, 3, 4, 5],[3, 4, 5, 6]])
>>> print(x[:2])
[[1 2 3 4]
 [2 3 4 5]]
```

实例 4：取得索引是 1 以后的 row 的元素。

```
>>> import numpy as np
>>> x = np.array([[1, 2, 3, 4],[2, 3, 4, 5],[3, 4, 5, 6]])
>>> print(x[1:])
[[2 3 4 5]
 [3 4 5 6]]
```

23-4-6　更改数组外形

reshape(row, column) 方法可以更改数组的维度。

实例 1：将一维数组转成二维 2×3 数组，然后将 2×3 数组转成 3×2 数组。

```
>>> import numpy as np
>>> x = np.array([1, 2, 3, 4, 5, 6])
>>> y = x.reshape(2, 3)
>>> print(y)
[[1 2 3]
 [4 5 6]]
>>> z = y.reshape(3, 2)
>>> print(z)
[[1 2]
 [3 4]
 [5 6]]
```

ravel( ) 可以将多维数组转成一维数组。

实例 2：将 2×3 数组转成一维数组。

```
>>> import numpy as np
>>> x = np.array([[1, 2, 3], [4, 5, 6]])
>>> y = x.ravel()
>>> print(y)
[1 2 3 4 5 6]
```

上述使用 reshape( ) 与 ravel( ) 方法执行数组外形更改时，不会更改原数组外形。如果使用 resize(row, column) 方法，则可以更改数组外形。

实例 3：二维 2×3 数组改为 3×2 数组，同时观察原数组外形。

```
>>> import numpy as np
>>> x = np.array([[1, 2, 3], [4, 5, 6]])
>>> x.resize(3, 2)
>>> print(x)
[[1 2]
 [3 4]
 [5 6]]
```

23-4-7　转置矩阵

所谓的转置矩阵是指将 n×m 矩阵转成 m×n 矩阵，transpose( ) 可以执行矩阵的转置。transpose( ) 也可以使用 T 取代，执行矩阵转置。

实例 1：矩阵转置的应用。

```
>>> import numpy as np
>>> x = np.arange(8).reshape(4, 2)
>>> print(x)
[[0 1]
 [2 3]
 [4 5]
 [6 7]]
>>> y = x.transpose()
>>> print(y)
[[0 2 4 6]
 [1 3 5 7]]
```

实例 2：使用 T 执行矩阵转置。

```
>>> import numpy as np
>>> x = np.arange(8).reshape(4,2)
>>> y = x.T
>>> print(y)
[[0 2 4 6]
 [1 3 5 7]]
```

23-4-8 将数组分割成子数组

hsplit( ) 可以将数组依水平方向分割，vsplit( ) 可以将数组依垂直方向分割。经此分割所返回的数组以列表方式存在。

实例 1：使用 hsplit( ) 方法依水平方向分割数组为 2 个子数组。

```
>>> import numpy as np
>>> x = np.arange(16).reshape(4, 4)
>>> print(x)
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]
 [12 13 14 15]]
>>> y1, y2 = np.hsplit(x,2)
>>> print(y1)
[[ 0  1]
 [ 4  5]
 [ 8  9]
 [12 13]]
>>> print(y2)
[[ 2  3]
 [ 6  7]
 [10 11]
 [14 15]]
```

实例 2：使用 vsplit( ) 方法依垂直方向分割数组为 2 个子数组。

```
>>> import numpy as np
>>> x = np.arange(16).reshape(4, 4)
>>> y1, y2 = np.vsplit(x,2)
>>> print(y1)
[[0 1 2 3]
 [4 5 6 7]]
>>> print(y2)
[[ 8  9 10 11]
 [12 13 14 15]]
```

23-4-9 矩阵堆栈

hstack( ) 可以执行矩阵水平方向堆栈，vstack( ) 可以执行矩阵垂直方向堆栈。column_stack( ) 可以将一维数组依 column 方向堆栈到二维数组，row_stack( ) 可以将一维数组依 row 方向堆栈到二维数组。

实例 1：使用 hstack( ) 执行数组依水平方向堆栈。

```
>>> import numpy as np
>>> x = np.arange(4).reshape(2,2)
>>> y = np.arange(4,8).reshape(2,2)
>>> z = np.hstack((x,y))
>>> print(x)
[[0 1]
 [2 3]]
>>> print(y)
[[4 5]
 [6 7]]
>>> print(z)
[[0 1 4 5]
 [2 3 6 7]]
```

实例 2：使用 vstack( ) 执行数组依垂直方向堆栈。

```
>>> import numpy as np
>>> x = np.arange(4).reshape(2,2)
>>> y = np.arange(4,8).reshape(2,2)
>>> z = np.vstack((x,y))
>>> print(z)
[[0 1]
 [2 3]
 [4 5]
 [6 7]]
```

实例 3：使用 column_stack( ) 将一维数组依 column 方向堆栈到二维数组。

```
>>> import numpy as np
>>> x = np.arange(4).reshape(2,2)
>>> y = np.array([5,6])
>>> z = np.column_stack((x,y))
>>> print(z)
[[0 1 5]
 [2 3 6]]
```

实例 4：使用 row_stack( ) 将一维数组依 row 方向堆栈到二维数组。

```
>>> import numpy as np
>>> x = np.arange(4).reshape(2,2)
>>> y = np.array([5,6])
>>> z = np.row_stack((x,y))
>>> print(z)
[[0 1]
 [2 3]
 [5 6]]
```

23-4-10　二维数组矩阵乘法运算

本节所述的矩阵乘法与线性代数的矩阵乘法意义相同，假设有一个 A 矩阵是 i×j 的二维数组，B 矩阵是 j×k 的二维数组，则 A 矩阵与 B 矩阵相乘可以得到 AB 矩阵是 i×k 的二维数组。

AB 矩阵的 AB_{ij} 值相当于是 A 矩阵的第 i 行乘以 B 矩阵的第 j 列，相当于 23-3-9 节所介绍的向量内积（inner product）。

$$ab_{ij} = \sum_{j=0}^{j-1} a_{ij} * b_{jk}$$

可以这样思考上述公式：

$$ab_{ij} = [\, a_{i0} \quad a_{i1} \quad \ldots \quad a_{i(j-1)} \,] * \begin{bmatrix} b_{0k} \\ b_{1k} \\ \vdots \\ b_{(j-1)k} \end{bmatrix}$$

$$= a_{i0} * b_{0k} + a_{i1} * b_{1k} + \ldots + a_{i(j-1)} * b_{(j-1)j}$$

下列是以数学领域的观点思考矩阵相乘，在数学领域矩阵左上角索引是（1,1）。

$$\mathbf{A} = \begin{bmatrix} a_{1,1} & a_{1,2} & \cdots \\ a_{2,1} & a_{2,2} & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix} \mathbf{B} = \begin{bmatrix} b_{1,1} & b_{1,2} & \cdots \\ b_{2,1} & b_{2,2} & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix}$$

$$\mathbf{AB} = \begin{bmatrix} a_{1,1}[b_{1,1} \quad b_{1,2} \quad \ldots] + a_{1,2}[b_{2,1} \quad b_{2,2} \quad \ldots] + \cdots \\ a_{2,1}[b_{1,1} \quad b_{1,2} \quad \ldots] + a_{2,2}[b_{2,1} \quad b_{2,2} \quad \ldots] + \cdots \\ \vdots \end{bmatrix}$$

矩阵乘法可以使用 dot( ) 或是 @ 运算符。

实例 1：使用 dot( ) 方法执行 2 个 2×2 的矩阵乘法运算。

```
>>> import numpy as np
>>> x = np.array([[1,2],[3,4]])
>>> y = np.array([[5,6],[7,8]])
>>> z = np.dot(x,y)
>>> print(z)
[[19 22]
 [43 50]]
```

实例 2：使用 @ 运算符执行 2×3 和 3×2 的矩阵乘法运算。

```
>>> import numpy as np
>>> x = np.array([[1,0,2],[-1,3,1]])
>>> y = np.array([[3,1],[2,1],[1,0]])
>>> z = x @ y
>>> print(z)
[[5 1]
 [4 2]]
```

23-4-11 将迭代运算应用在二维数组

程序实例 ch23_2.py：建立一个 1 ～ 100 的 10×10 数组，然后使用迭代做加总运算。

```
1  # ch23_2.py
2  import numpy as np
3
4  A = 0
5  X = np.arange(1,101).reshape(10,10)
6  print(X)
7  for x in X:
8      A += x
9  print(type(A))
10 print("A = ", A)
11
12 sum = 0
13 for a in A:
14     sum += a
15 print(type(sum))
16 print("sum = ", sum)
```

执行结果

```
===================== RESTART: D:/Python/ch23/ch23_2.py =====================
[[  1   2   3   4   5   6   7   8   9  10]
 [ 11  12  13  14  15  16  17  18  19  20]
 [ 21  22  23  24  25  26  27  28  29  30]
 [ 31  32  33  34  35  36  37  38  39  40]
 [ 41  42  43  44  45  46  47  48  49  50]
 [ 51  52  53  54  55  56  57  58  59  60]
 [ 61  62  63  64  65  66  67  68  69  70]
 [ 71  72  73  74  75  76  77  78  79  80]
 [ 81  82  83  84  85  86  87  88  89  90]
 [ 91  92  93  94  95  96  97  98  99 100]]
<class 'numpy.ndarray'>
A =  [460 470 480 490 500 510 520 530 540 550]
<class 'numpy.int32'>
sum =  5050
```

上述第 7 行的 x 是数组 X 的元素，其实是一个子数组，所以所得到的 A 也是数组。第 13 行的 a 则是 A 数组的元素，它是 32 位整数，所以最后可以得到总和。

23-5 简单线性代数运算

23-5-1 一元二次方程式

一元二次方程式的概念可以参考 5-8-4 节，Numpy 有 roots( ) 方法可以解一元二次方程式的根，假设有一个方程式如下：

$$ax^2 + bx + c = 0$$

可以直接带入 roots（[a, b, c]）即可求解。

实例：求 $3x^2 + 5x + 1 = 0$ 的根。

```
>>> import numpy as np
>>> r = np.roots([3,5,1])
>>> print(r)
[-1.43425855 -0.23240812]
```

可以得到与程序实例 ch5_13.py 相同结果。

23-5-2　解联立线性方程式

使用 Numpy 可以处理线性代数的问题。假设有两个线性方程式如下：

3x + 5y = 18

2x + 3y = 11

我们可以建立两个数组储存上述方程式，一个是 x 和 y 的系数数组，另一个是方程式右边值的因变量数组。

然后可以使用 linalg 模块的 solve( ) 函数，最后可以得到下列 x=1 和 y=3。

```
>>> import numpy as np
>>> coeff = np.array([[3,5],[2,3]])
>>> deps = np.array([18,11])
>>> ans = np.linalg.solve(coeff, deps)
>>> print(ans)
[1. 3.]
```

下列是验证这个结果，其中 10.999……是浮点数的问题，可视为是 11。

```
>>> print(3*ans[0] + 5*ans[1])
18.0
>>> print(2*ans[0] + 3*ans[1])
10.999999999999998
```

我们也可以使用内积方式验证此结果：

```
>>> y = np.dot(coeff, ans)
>>> print(y)
[18. 11.]
```

如果上述计算正确，上述 y 将很接近 deps 的数组值，因为可能有浮点数舍去的问题。我们也可以用 allclose( ) 验证此计算：

```
>>> np.allclose(y, deps)
True
```

23-6　Numpy 的广播功能

Numpy 在执行两个数组运算时，原则上数组外形必须兼容才可运算，如果外形不同 Numpy 可以使用广播（broadcast）机制，先将比较小的数组扩大至与较大的数组外形相同，然后再执行运算。

实例 1：将整数 5 或数组 [5] 与数组 [1,2,3] 相加。

```
>>> import numpy as np
>>> x = np.array([1,2,3])
>>> y = 5
>>> z = x + y
>>> print(z)
[6 7 8]
>>> r = [5]
>>> s = x + r
>>> print(s)
[6 7 8]
```

其实对上述实例而言，不论是整数 5 或数组 [5]，与数组 [1,2,3] 相加时，皆会先被扩张为（3,）的数组 [5,5,5]，然后再执行运算。

假设有一个（3,）之一维数组，另有一个（2,3）之二维数组，则（3,）之一维数组会先被扩张为（2,3）之二维数组然后执行运算。

实例 2：将（3,）之一维数组 [1,2,3] 与（2,3）之二维数组 [[1,2,3],[4,5,6]] 相加。

```
>>> import numpy as np
>>> x = np.array([1,2,3])
>>> y = np.array([[1,2,3],[4,5,6]])
>>> z = x + y
>>> print(z)
[[2 4 6]
 [5 7 9]]
```

其实上述是 Numpy 先将 [1,2,3] 扩张为 [[1,2,3],[1,2,3]]，然后才执行运算。

实例 3：两个数组皆扩张的应用。

```
>>> import numpy as np
>>> x = np.array([1,2,3]).reshape(3,1)
>>> print(x)
[[1]
 [2]
 [3]]
>>> y = np.ones(5)
>>> print(y)
[1. 1. 1. 1. 1.]
>>> z = x + y
>>> print(z)
[[2. 2. 2. 2. 2.]
 [3. 3. 3. 3. 3.]
 [4. 4. 4. 4. 4.]]
```

上述相当于在执行 x+y 时，x 会扩张为：

```
[[1 1 1 1 1]
 [2 2 2 2 2]
 [3 3 3 3 3]]
```

y 会扩张为：

```
[[1., 1., 1., 1., 1.],
 [1., 1., 1., 1., 1.],
 [1., 1., 1., 1., 1.]]
```

所以可以得到上述 z 的执行结果。

其实并不是所有数组运算皆可以扩张数组，例如（2,）之一维数组就无法扩张与（3,）之一维数组执行运算。

实例 4：（2,）之一维数组与（3,）之一维数组执行加法运算，产生错误的实例。

```
>>> import numpy as np
>>> x = np.array([1,2])
>>> y = np.array([1,2,3])
>>> z = x + y
Traceback (most recent call last):
  File "<pyshell#511>", line 1, in <module>
    z = x + y
ValueError: operands could not be broadcast together with shapes (2,) (3,)
```

23-7 常用的数学函数

更完整的 Numpy 模块的数学方法可以参考下列网址：

https://docs.scipy.org/doc/numpy/reference/routines.math.html

23-7-1 三角函数相关知识

除了常见的 sin(x)、cos(x)、tan(x)、arcsin(x)、arccos(x)、arctan(x) 外，下列是比较特别的函数：

degrees(x)：将弧度（radians）转成角度。

radians(x)：将角度转成弧度（radians）。

实例 1：将数组弧度转成角度。

```
>>> import numpy as np
>>> rad = np.arange(12)*np.pi/6
>>> x = np.degrees(rad)
>>> print(x)
[  0.  30.  60.  90. 120. 150. 180. 210. 240. 270. 300. 330.]
```

实例 2：将数组角度转成弧度。

```
>>> import numpy as np
>>> deg = np.arange(12)*30
>>> x = np.radians(deg)
>>> print(x)
[0.         0.52359878 1.04719755 1.57079633 2.0943951  2.61799388
 3.14159265 3.66519143 4.1887902  4.71238898 5.23598776 5.75958653]
```

23-7-2　和 sum( )、积 prod( )、差 diff( ) 函数

下列是常见的函数：

prod(a, axis=None)：返回指定轴（axis）的数组 a 元素的乘积。

实例 1：如果是空数组，结果是 1.0。

```
>>> np.prod([])
1.0
```

实例 2：如果是一维数组，则是元素的乘积。

```
>>> np.prod([1,2,3])
6
```

实例 3：如果是二维数组，也是返回所有元素的乘积。

```
>>> np.prod([[1,2],[3,4]])
24
```

实例 4：返回指定轴的元素乘积。

```
>>> np.prod([[1,2],[3,4]], axis=1)
array([ 2, 12])
```

sum(a, axis=None)：返回指定轴（axis）的数组 a 元素的总和。

实例 5：如果是空数组，结果是 0.0。

```
>>> np.sum([])
0.0
```

实例 6：如果是一维数组，则是元素的加总。

```
>>> np.sum([1,2,3])
6
```

实例 7：元素是浮点数，但是设置数据是 int32。

```
>>> np.sum([1.2,1.5,3.1],dtype=np.int32)
5
```

实例 8：使用不同轴，执行二维数组元素的加总。

```
>>> np.sum([[1,2],[3,4]])
10
>>> np.sum([[1,2],[3,4]], axis=0)
array([4, 6])
>>> np.sum([[1,2],[3,4]], axis=1)
array([3, 7])
```

程序实例 ch23_3.py：使用 sum( ) 函数重新设计 ch23_2.py。

```
1  # ch23_3.py
2  import numpy as np
3
4  X = np.arange(1,101).reshape(10,10)
5  A = np.sum(X, axis=0)
6  print("A = ", A)
7  sum = np.sum(X)
8  print("sum = ", sum)
```

执行结果

```
==================== RESTART: D:/Python/ch23/ch23_3.py ====================
A =  [460 470 480 490 500 510 520 530 540 550]
sum =  5050
```

diff(a, n, axis)：返回指定轴的元素差（后一个元素值减去前一个元素值），n 代表执行几次。

实例 9：一维数组执行 1 次与执行 2 次的结果。

```
>>> x = np.array([1, 4, 7, 0, 5])
>>> np.diff(x)
array([ 3,  3, -7,  5])
>>> np.diff(x, n=2)
array([  0, -10,  12])
```

实例 10：使用不同轴，执行二维数组元素差的计算。

```
>>> x = np.array([[1, 4, 6, 10], [0, 2, 5, 9]])
>>> np.diff(x)
array([[3, 2, 4],
       [2, 3, 4]])
>>> np.diff(x, axis=0)
array([[-1, -2, -1, -1]])
```

23-7-3 舍去函数

around(a, decimals=0)：可以舍至最接近的偶数整数，decimals 则是指定小数位数。

实例 1：系列数组的 around( ) 操作。

```
>>> np.around([0.49, 1.82])
array([0., 2.])
>>> np.around([0.49, 1.82], decimals=1)
array([0.5, 1.8])
>>> np.around([0.5, 1.5, 2.5, 3.5, 4.5, 5.4])
array([0., 2., 2., 4., 4., 5.])
```

rint(x)：返回最接近的整数。

实例 2：系列数组元素的 rint( ) 操作。

```
>>> np.rint([1.4, 1.5, 1.6, 2.5])
array([1., 2., 2., 2.])
```

floor(x)：返回小于或等于对象的最大整数。

实例 3：系列数组元素的 floor( ) 运作。

```
>>> np.floor([-1.5, 0.8, 1.2])
array([-2.,  0.,  1.])
```

ceil(x)：返回大于或等于对象的最小整数。

实例 4：系列数组元素的 ceil( ) 运作。

```
>>> np.ceil([-1.5, 0.8, 1.2])
array([-1.,  1.,  2.])
```

trunc(x)：舍去小数的 trunc( ) 操作。

实例 5：系列数组元素小数的 trunc( ) 操作。

```
>>> np.trunc([-1.3, -2.8, 0.5, 2.9])
array([-1., -2.,  0.,  2.])
```

23-7-4　最大公因子与最小公倍数

gcd(x)：返回数组元素的最大公因子（greatest common divisor）。

实例 1：最大公因子 gcd( ) 的应用。

```
>>> np.gcd(12, 20)
4
>>> np.gcd.reduce([15, 35, 55])
5
```

lcm(x1, x2)：返回数组元素的最小公倍数（lowest common multiple）。

实例 2：最小公倍数 lcm( ) 的应用。

```
>>> np.lcm(12, 20)
60
>>> np.lcm.reduce([6, 12, 60])
60
```

23-7-5　指数与对数

exp(x)：返回数组元素 x 自然对数 e 的次方。

实例 1：exp( ) 的应用。

```
>>> np.exp([1,2,3])
array([ 2.71828183,  7.3890561 , 20.08553692])
```

exp2(x)：返回数组元素 x 的 2 的次方。

实例 2：exp2( ) 的应用。

```
>>> np.exp2([1,2,3])
array([2., 4., 8.])
```

log(x)：返回数组元素 x 的自然对数值。

实例 3：log( ) 的应用。

```
>>> np.log([1, np.e, np.e**2, 0])
array([  0.,   1.,   2., -inf])
```

log2(x)：返回数组元素 x 的自然对数值。

实例 4：log2( ) 的应用。

```
>>> np.log2([0, 1, 2, 2**5])
array([-inf,   0.,   1.,   5.])
```

log10(x)：返回数组元素 x 的自然对数值。

实例 5：log10( ) 的应用。

```
>>> np.log10([10, 1000, 5])
array([1.      , 3.      , 0.69897])
```

23-7-6 算术运算

add(x1, x2)：相当于“+”加法运算。

subtract(x1, x2)：相当于“-”减法运算。

multiply(x1, x2)：相当于“*”乘法运算。

divide(x1, x2)：相当于“/”除法运算。

mod(x1, x2)：相当于“%”求余数运算。

remainder(x1, x2)：相当于“%”求余数运算。

negative(x1)：相当于正号变为负号，负号变为正号。

实例 1：negative( ) 的应用。

```
>>> np.negative([1, -1])
array([-1,  1])
```

divmod(x1, x2)：x1 除以 x2，返回商与余数，返回是含 2 个元素的元组（tuple），第 1 个元素是商，第 2 个元素是余数。

实例 2：divmod( ) 的应用。

```
>>> np.divmod(np.arange(5), 2)
(array([0, 0, 1, 1, 2], dtype=int32), array([0, 1, 0, 1, 0], dtype=int32))
```

23-7-7 其他函数

absolute(x)：返回绝对值。

实例 1：absolute( ) 的应用。

```
>>> np.negative([-3, 3])
array([ 3, -3])
```

square(x)：返回平方值。

实例 2：square( ) 的应用。

```
>>> np.square([1, 3])
array([1, 9], dtype=int32)
```

sqrt(x)：返回平方根。

实例 3：sqrt( ) 的应用。

```
>>> np.sqrt([1, 4, 9, 15])
array([1.        , 2.        , 3.        , 3.87298335])
```

sign(x)：小于 0 返回 -1，等于 0 返回 0，大于 0 返回 1。

实例 4：sign( ) 的应用。

```
>>> np.sign([-1, -0.5,  0, 0.5, 1])
array([-1., -1.,  0.,  1.,  1.])
```

max(x)：返回数组最大元素。

实例 5：max( ) 的应用。

```
>>> np.max([1,2,3])
3
>>> np.max(np.arange(100).reshape(10,10))
99
```

maximum(x1, x2)：返回数组中相同位置较大的元素值。

实例 6：maximum( ) 的应用。

```
>>> np.maximum([1, 5, 10], [3, 4, 9])
array([ 3,  5, 10])
```

min(x)：返回数组最小元素。

实例 7：min( ) 的应用。

```
>>> np.min([1,2,3])
1
>>> np.min(np.arange(100).reshape(10,10))
0
```

minimum(x1, x2)：返回数组中相同位置较小的元素值。

实例 8：minimum( ) 的应用。

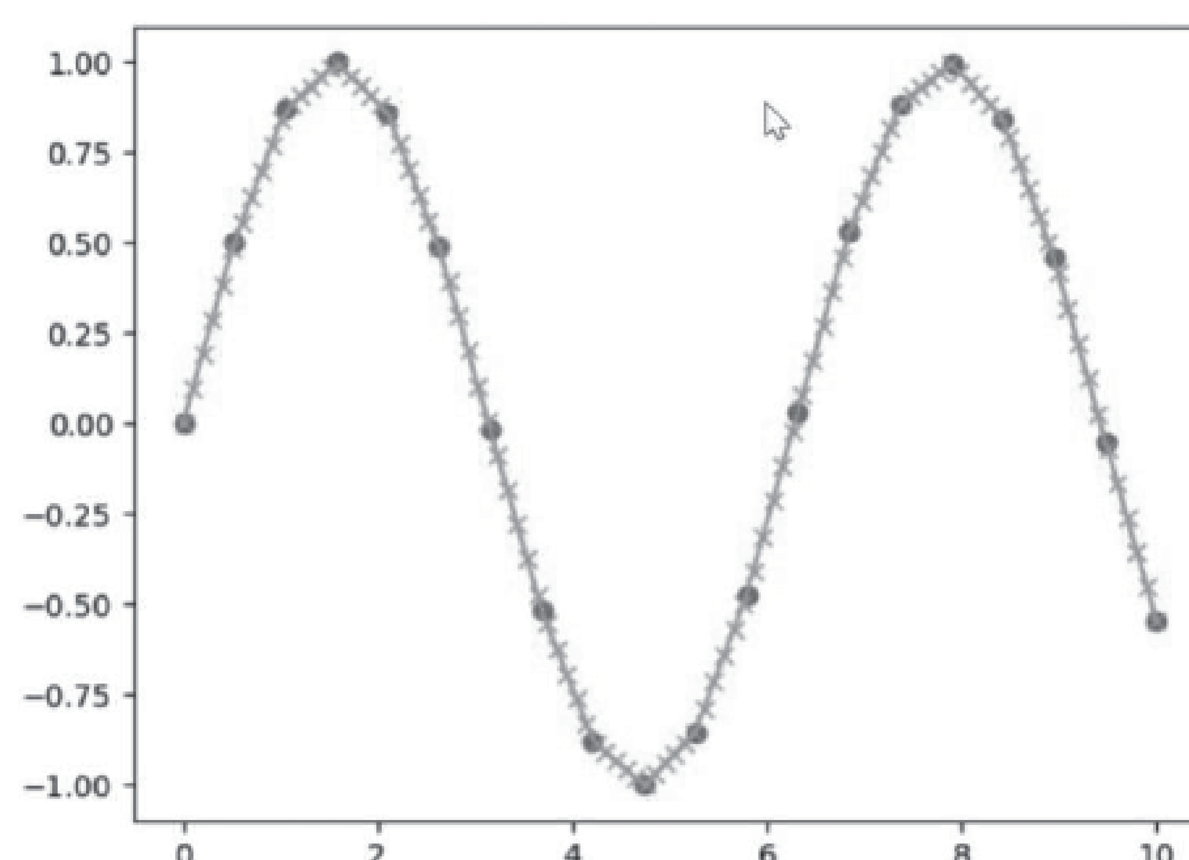
```
>>> np.minimum([1, 5, 10], [3, 4, 9])
array([1, 4, 9])
```

interp(x, xp, fp)：一维数组的线性插入，xp 是 x 轴的坐标，yp 是 y 轴的坐标，x 则是 x 轴的插入值，然后可以由此计算出 y 轴的值。

程序实例 ch23_4.py：线性插入 interp( ) 的应用，这个程序会在 x 轴 0 ～ 10 之间建立均分的 20 个点，这些点用 o 做标记然后是依 sin(x) 计算相对应的 y 轴值，然后采用 interp( ) 插入 100 个点，这 100 个点使用 x 标记，同时将 100 点连接。

```
1  # ch23_4.py
2  import numpy as np
3  import matplotlib.pyplot as plt
4
5  x = np.linspace(0, 10, 20)
6  y = np.sin(x)
7
8  xvals = np.linspace(0, 10, 100)
9  yinterp = np.interp(xvals, x, y)
10
11 plt.plot(x, y, 'o')
12 plt.plot(xvals, yinterp, '-x')
13 plt.show()
```

执行结果



23-8 随机数函数

更完整的 Numpy 模块的随机数函数可以参考下列网址：

https://docs.scipy.org/doc/numpy/reference/routines.random.html

23-8-1 简单随机数据

rand(d0, d1, … dn)：返回指定外形的数组元素，值在 [0, 1）间，[0, 1）表示含 0 不含 1。由于是随机，所以每次执行结果皆不相同。

实例 1：rand( ) 的应用。

```
>>> np.random.rand(3)
array([0.47164429, 0.82153141, 0.41865045])
>>> np.random.rand(3,2)
array([[0.74758203, 0.13709832],
       [0.97030083, 0.7928294 ],
       [0.34886091, 0.4641032 ]])
```

randn(d0, d1, … dn)：所返回的随机数是标准常态分布（standard normal distribution），0 是均值，1 是标准偏差的正态分布。

实例 2：randn( ) 的应用。

```
>>> np.random.randn()
0.86833652406693
>>> np.random.randn(2, 3)
array([[-0.34099598,  0.24438972,  0.56923048],
       [-1.05048661, -0.00602095,  3.55042135]])
```

randint(low[,high, size, dtype])：返回介于 low 和 high 之间的随机整数 [low, high)，包含 low 不包含 high。如果省略 high，则所产生的随机整数在 [0, low）间。

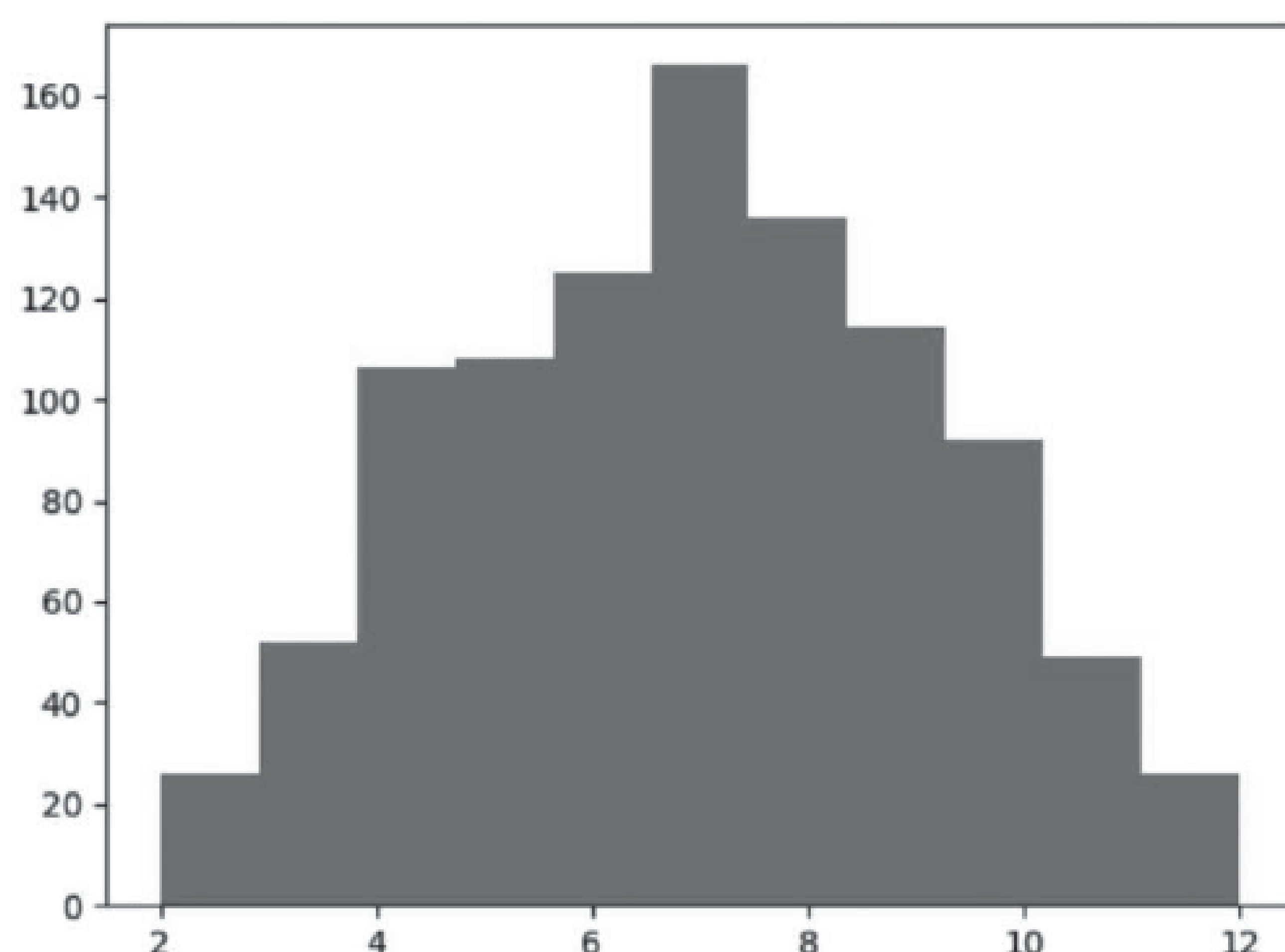
实例 3：randint( ) 的应用。

```
>>> np.random.randint(5)
0
>>> np.random.randint(0, 10, size=5)
array([9, 6, 8, 4, 9])
```

程序实例 ch23_5.py：骰子 2 颗各掷 1000 次，然后以直方图列出 2 颗加总所产生数值的直方图。

```
1  # ch23_5.py
2  import numpy as np
3  import matplotlib.pyplot as plt
4
5  d1 = np.random.randint(1,6+1,1000)
6  d2 = np.random.randint(1,6+1,1000)
7  dsums = d1 + d2
8
9  plt.hist(dsums, bins=11)
10 plt.show()
```

执行结果



random_integers(low,[,high,size])：返回介于 low 和 high 之间的随机整数 [low, high]，包含 low 也包含 high。如果省略 high，则所产生的随机整数在 [1, low] 间。其实建议可以使用 randint( ) 取代此函数。

实例 4：random_integers( ) 的应用。

```
>>> np.random.random_integers(5)
3
>>> np.random.random_integers(5, size=(2,3))
array([[3, 5, 2],
       [2, 1, 1]])
```

choice(a[,size=None,replace=True,p=None])：从指定数组中随机返回元素，如果 a 是整数，相当于是 np.arange(a)，size 是返回数量。

实例 5：choice( ) 的应用。

```
>>> np.random.choice([1,2,3,4,5],3)
array([5, 5, 3])
>>> np.random.choice(6,3)
array([3, 1, 5])
```

23-8-2　顺序变更

shuffle(x)：将数组元素位置随机重新排列。

实例 1：shuffle( ) 的应用。

```
>>> x = np.arange(10)
>>> np.random.shuffle(x)
>>> x
array([2, 6, 7, 4, 5, 3, 9, 1, 8, 0])
>>> y = np.arange(9).reshape(3,3)
>>> np.random.shuffle(y)
>>> y
array([[3, 4, 5],
       [0, 1, 2],
       [6, 7, 8]])
```

permutation(x)：返回随机重排元素的数组，原数组元素位置没有更改。如果 x 是整数，相当于是 np.arange(x)。

实例 2：permutation(x) 的应用。

```
>>> x = np.arange(9)
>>> y = np.random.permutation(x)
>>> print(y)
[6 3 5 1 2 7 8 4 0]
>>> a = np.arange(15).reshape(3,5)
>>> b = np.random.permutation(a)
>>> print(b)
[[ 0  1  2  3  4]
 [10 11 12 13 14]
 [ 5  6  7  8  9]]
>>> np.random.permutation(10)
array([2, 3, 4, 1, 8, 9, 5, 7, 6, 0])
```

23-8-3　分布

beta(a, b[,size])：Beta 分布取样。

binomial(n, p[,size])：二项分布取样。

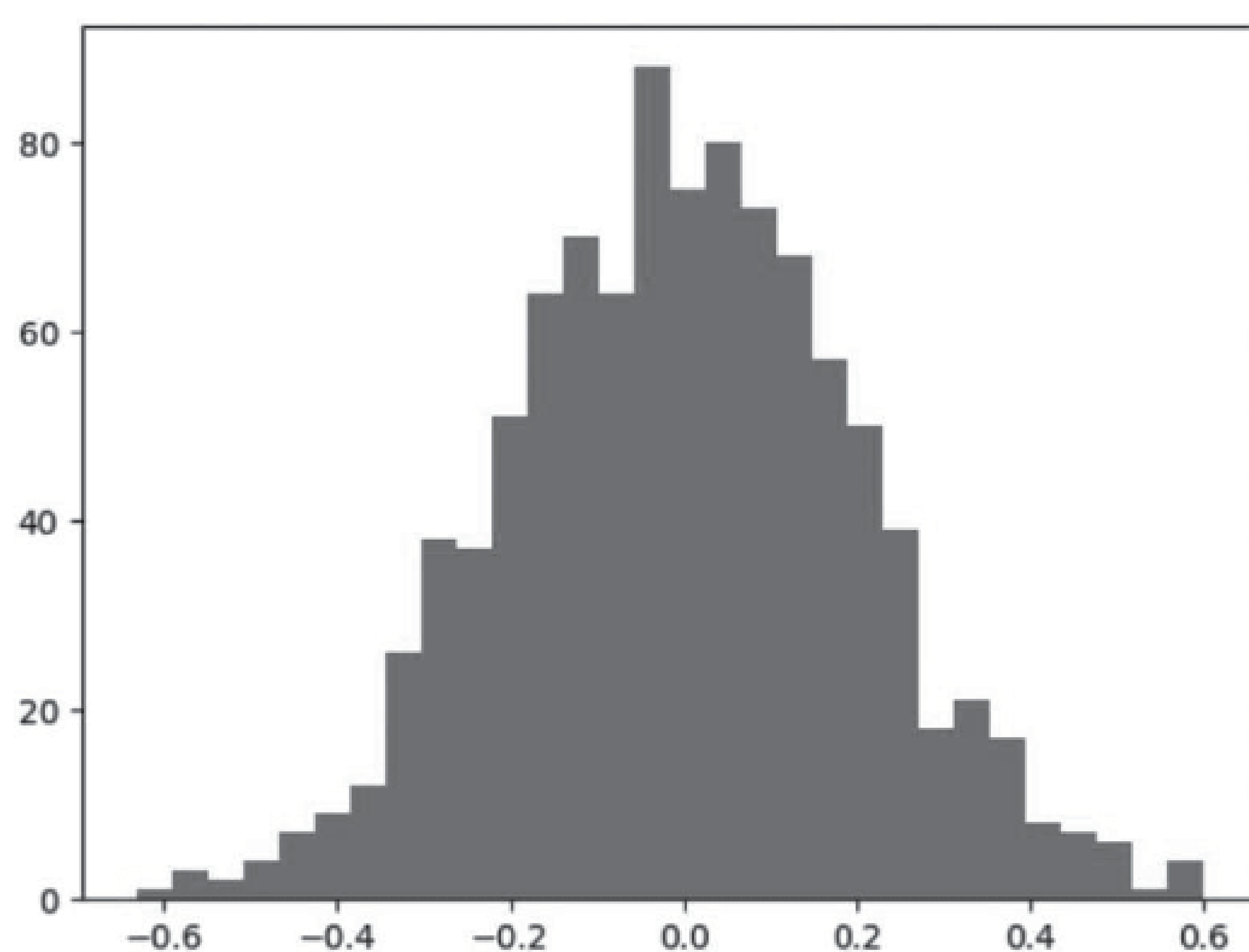
chisquare(df[,size])：卡方（chi-square）分布取样。

normal([loc, scale, size])：从常态分布取样。loc 是平均值（mean），scale 是标准偏差（standard deviation），size 是样本数。

程序实例 ch23_6.py：normal( ) 的应用，绘制常态分布，bins 数量是 30。

```
1  # ch23_6.py
2  import numpy as np
3  import matplotlib.pyplot as plt
4
5  mean, sigma = 0, 0.2
6  s = np.random.normal(mean, sigma, 1000)
7
8  plt.hist(s, bins=30)
9  plt.show()
```

执行结果

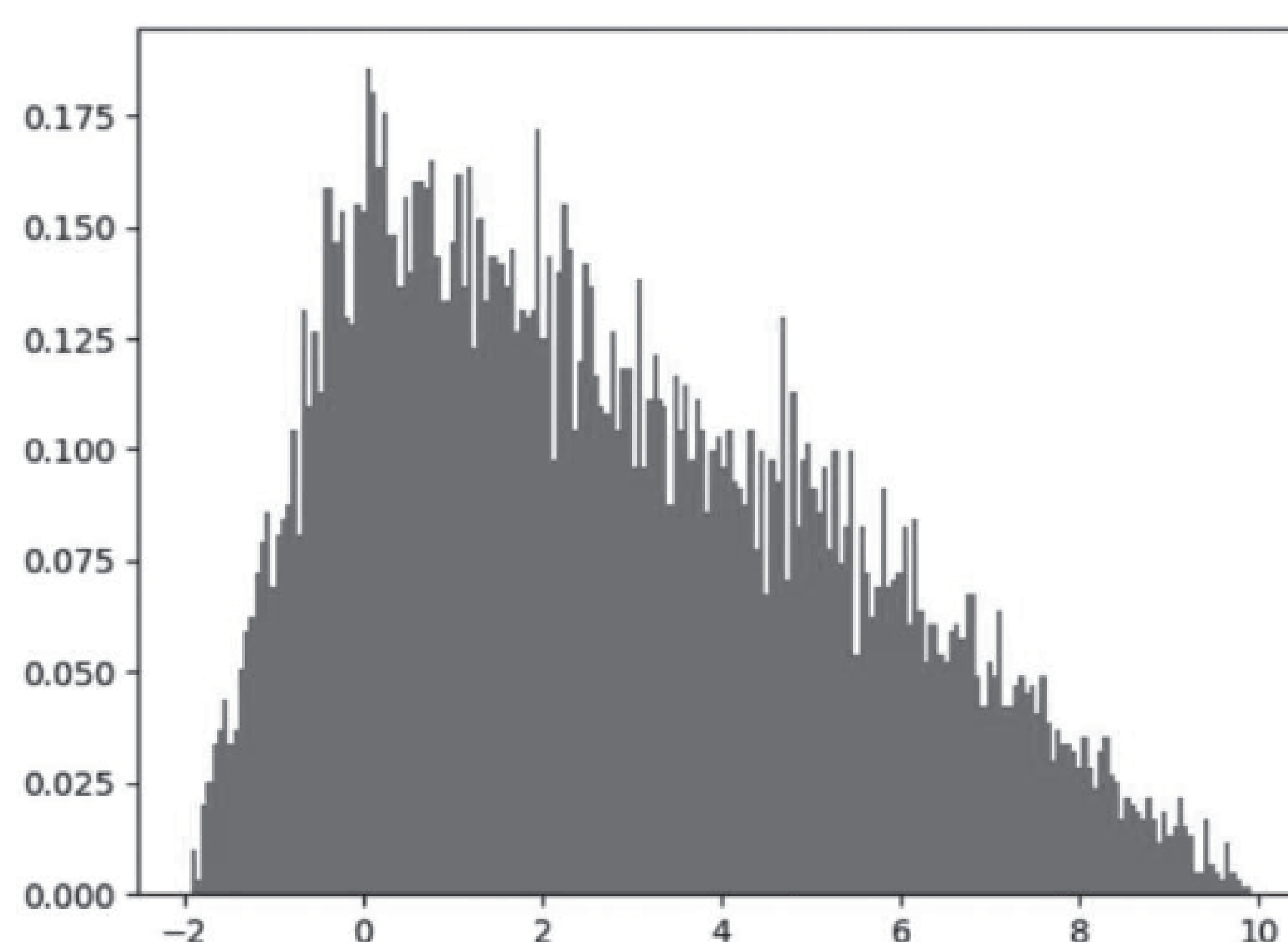


triangular(left, mode, right, size=None)：三角形分布取样，left 是最小值，mode 是尖峰值，right 是最大值，size 是样本数。

程序实例 ch23_7.py：三角形分布取样的实例，这个程序在呼叫 hist( ) 方法时，增加设置 density=True，此时 y 轴不再是次数，而是概率值。

```
1  # ch23_7.py
2  import numpy as np
3  import matplotlib.pyplot as plt
4
5  s = np.random.triangular(-2, 0, 10, 10000)
6  plt.hist(s, bins=200, density=True)
7  plt.show()
```

执行结果



更多细节读者可参考 23-8 节提到的网站。

23-9 统计函数

更完整的 Numpy 模块的统计函数可以参考下列网址：

https://docs.scipy.org/doc/numpy/reference/routines.statistics.html

23-9-1 统计

amin(a[,axis])：返回数组最小元素或是指定轴的最小元素。

nanmin(a[,axis])：返回数组最小元素或是指定轴的最小元素，忽略 NaN。

实例 1：amin( ) 的应用。

```
>>> x = np.arange(4).reshape((2,2))
>>> np.amin(x)
0
>>> np.amin(x, axis=0)
array([0, 1])
>>> np.amin(x, axis=1)
array([0, 2])
```

amax(a[,axis])：返回数组最大元素或是指定轴的最大元素。

nanmax(a[,axis])：返回数组最大元素或是指定轴的最大元素，忽略 NaN。

实例 2：amax( ) 的应用。

```
>>> x = np.arange(4).reshape((2,2))
>>> np.amax(x)
3
>>> np.amax(x, axis=0)
array([2, 3])
>>> np.amax(x, axis=1)
array([1, 3])
```

23-9-2 平均和变异数

在 8-15-2 节笔者有说明基础统计平均值、变异数、标准偏差的计算方式，学会本节，未来读者可以直接套用，省去许多时间。

average(a[,axis,weights])：如果省略 weights 返回数组的平均，如果有 weights 则返回数组的加权平均。

实例 1：average( ) 的应用。

```
>>> x = np.arange(1,5)
>>> np.average(x)
2.5
>>> np.average(x,weights=range(4,0,-1))
2.0
```

mean(a[,axis])：返回数组元素平均值或指定轴的数组元素平均值。

实例 2：mean( ) 的应用。

```
>>> x = np.array([[1,2],[3,4]])
>>> np.mean(x)
2.5
>>> np.mean(x, axis=0)
array([2., 3.])
>>> np.mean(x, axis=1)
array([1.5, 3.5])
```

median(a[,axis])：计算数组的中位数或指定轴的中位数。

实例 3：median( ) 的应用。

```
>>> x = np.array([[12,7,4],[3,2,6]])
>>> np.median(x)
5.0
>>> np.median(x, axis=0)
array([7.5, 4.5, 5. ])
>>> np.median(x, axis=1)
array([7., 3.])
```

std(a[,axis])：计算数组的标准偏差或指定轴的标准偏差。

实例 4：std( ) 的应用。

```
>>> x = np.arange(1,5).reshape(2,2)
>>> np.std(x)
1.118033988749895
>>> np.std(x, axis=0)
array([1., 1.])
>>> np.std(x, axis=1)
array([0.5, 0.5])
```

var(a[,axis])：计算数组的变异数或指定轴的变异数。

实例 5：var( ) 的应用。

```
>>> x = np.arange(1,5).reshape(2,2)
>>> np.var(x)
1.25
>>> np.var(x, axis=0)
array([1., 1.])
>>> np.var(x, axis=1)
array([0.25, 0.25])
```

23-10　文件的输入与输出

更多文件的输入与输出知识可以参考下列网址：

https://docs.scipy.org/doc/numpy/reference/routines.io.html

在真实的应用中，我们必须从文件读取数据，或是将数据写入文件，这些文件可能是文本文件（.txt 或 .csv）或是二进制文件，这将是本节的主题。

23-10-1　读取文本文件

Numpy 有提供 loadtxt( ) 可以执行读取文件，存入数组，它的语法如下：

```
loadtxt(fname, dtype=<class 'float'>, comments='#', delimiter=None, skiprows=0,
        usecols=None, encoding='bytes', … 其他参数 )
```

fname 文件名，dtype 数据形态，comments 文件注释，delimiter 分隔字符，skiprows 忽略前几 rows，usecols 读取那些 columns，encoding 文件编码。

程序实例 ch23_8.py：有一个 txt 文件内容如下，请读取此文件 ch23_8.txt，然后打印文件内的数组。

```
1,2,3,4,5,6,7,8,9
11,12,13,14,15,16,17,18,19
21,22,23,24,25,26,27,28,29
31,32,33,34,35,36,37,38,39
41,42,43,44,45,46,47,48,49
```

```
1  # ch23_8.py
2  import numpy as np
3
4  x = np.loadtxt("ch23_8.txt",delimiter=',')
5  print(x)
```

执行结果

```
===================== RESTART: D:/Python/ch23/ch23_8.py =====================
[[ 1.  2.  3.  4.  5.  6.  7.  8.  9.]
 [11. 12. 13. 14. 15. 16. 17. 18. 19.]
 [21. 22. 23. 24. 25. 26. 27. 28. 29.]
 [31. 32. 33. 34. 35. 36. 37. 38. 39.]
 [41. 42. 43. 44. 45. 46. 47. 48. 49.]]
```

程序实例 ch23_9.py：忽略前 2 row，只取第 (1, 3, 5)column，留意 column 是从 0 开始计数。

```
1  # ch23_9.py
2  import numpy as np
3
4  x = np.loadtxt("ch23_8.txt",delimiter=',', skiprows=2, usecols=(1,3,5))
5  print(x)
```

执行结果

```
===================== RESTART: D:/Python/ch23/ch23_9.py =====================
[[22. 24. 26.]
 [32. 34. 36.]
 [42. 44. 46.]]
```

23-10-2 写入文本文件

Numpy 有提供 savetxt( )，可以执行将数组写入文件，它的语法如下：

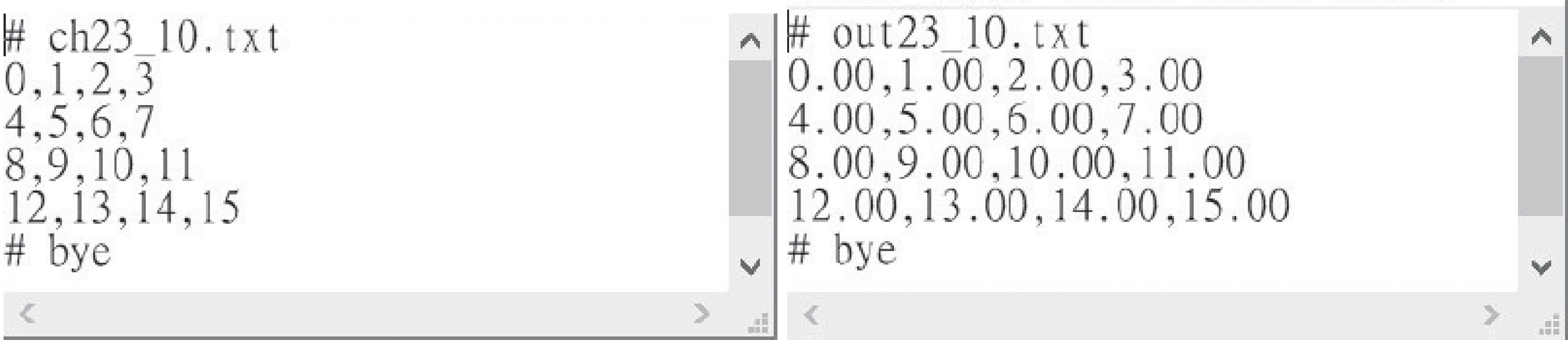
savetxt(fname, fmt='%.18e', comments='#', delimiter=None, header='', footer='', encoding='bytes', … 其他参数)

上述参数 header 是配置文件开头字符串，footer 是设置文件尾字符串，fmt 是格式化数据。

程序实例 ch23_10.py：写入数组数据。

```
1  # ch23_10.py
2  import numpy as np
3
4  x = np.arange(16).reshape(4,4)
5  np.savetxt('ch23_10.txt',x,delimiter=',', header='ch23_10.txt',
6             footer='bye',fmt="%d")
7  np.savetxt('out23_10.txt',x,delimiter=',', header='out23_10.txt',
8             footer='bye',fmt="%4.2f")
```

执行结果



```
# ch23_10.txt
0,1,2,3
4,5,6,7
8,9,10,11
12,13,14,15
# bye
```

```
# out23_10.txt
0.00,1.00,2.00,3.00
4.00,5.00,6.00,7.00
8.00,9.00,10.00,11.00
12.00,13.00,14.00,15.00
# bye
```

习题

1. 在 0 ～ 5 之间产生 20 个等距数组。(**20-3 节**)

```
====================== RESTART: D:/Python/ex/ex23_1.py ======================
[0.         0.26315789 0.52631579 0.78947368 1.05263158 1.31578947
 1.57894737 1.84210526 2.10526316 2.36842105 2.63157895 2.89473684
 3.15789474 3.42105263 3.68421053 3.94736842 4.21052632 4.47368421
 4.73684211 5.        ]
```

2. 请建立 1 ～ 50 的 5×10 矩阵。(**20-3 节**)

```
====================== RESTART: D:/Python/ex/ex23_2.py ======================
[[ 1  2  3  4  5  6  7  8  9 10]
 [11 12 13 14 15 16 17 18 19 20]
 [21 22 23 24 25 26 27 28 29 30]
 [31 32 33 34 35 36 37 38 39 40]
 [41 42 43 44 45 46 47 48 49 50]]
```

3. 请建立下列 2 个 A, B 矩阵。(**20-4 节**)

$$A = \begin{bmatrix} 2 & 2 \\ 2 & 2 \\ 2 & 2 \end{bmatrix} \qquad B = \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix}$$

然后分别列出加、减、乘、除、求余数以及 A 和 B 的转矩阵。

```
==================== RESTART: D:\Python\ex\ex23_3.py ====================
A    =
 [[2 2]
 [2 2]
 [2 2]]
B    =
 [[1 4]
 [2 5]
 [3 6]]
A+B =
 [[3 6]
 [4 7]
 [5 8]]
A-B =
 [[ 1 -2]
 [ 0 -3]
 [-1 -4]]
A*B =
 [[ 2  8]
 [ 4 10]
 [ 6 12]]
A/B =
 [[2.         0.5       ]
 [1.         0.4       ]
 [0.66666667 0.33333333]]
A%B =
 [[0 2]
 [0 2]
 [2 2]]
A转置 =
 [[2 2 2]
 [2 2 2]]
B转置 =
 [[1 2 3]
 [4 5 6]]
```

4. 请解下列方程式。（**20-4 节**）

6x + 5y = 100

9x + 2y = 50

```
====================== RESTART: D:/Python/ex/ex23_4.py ======================
x =  1.5151515151515156
y =  18.18181818181818
```

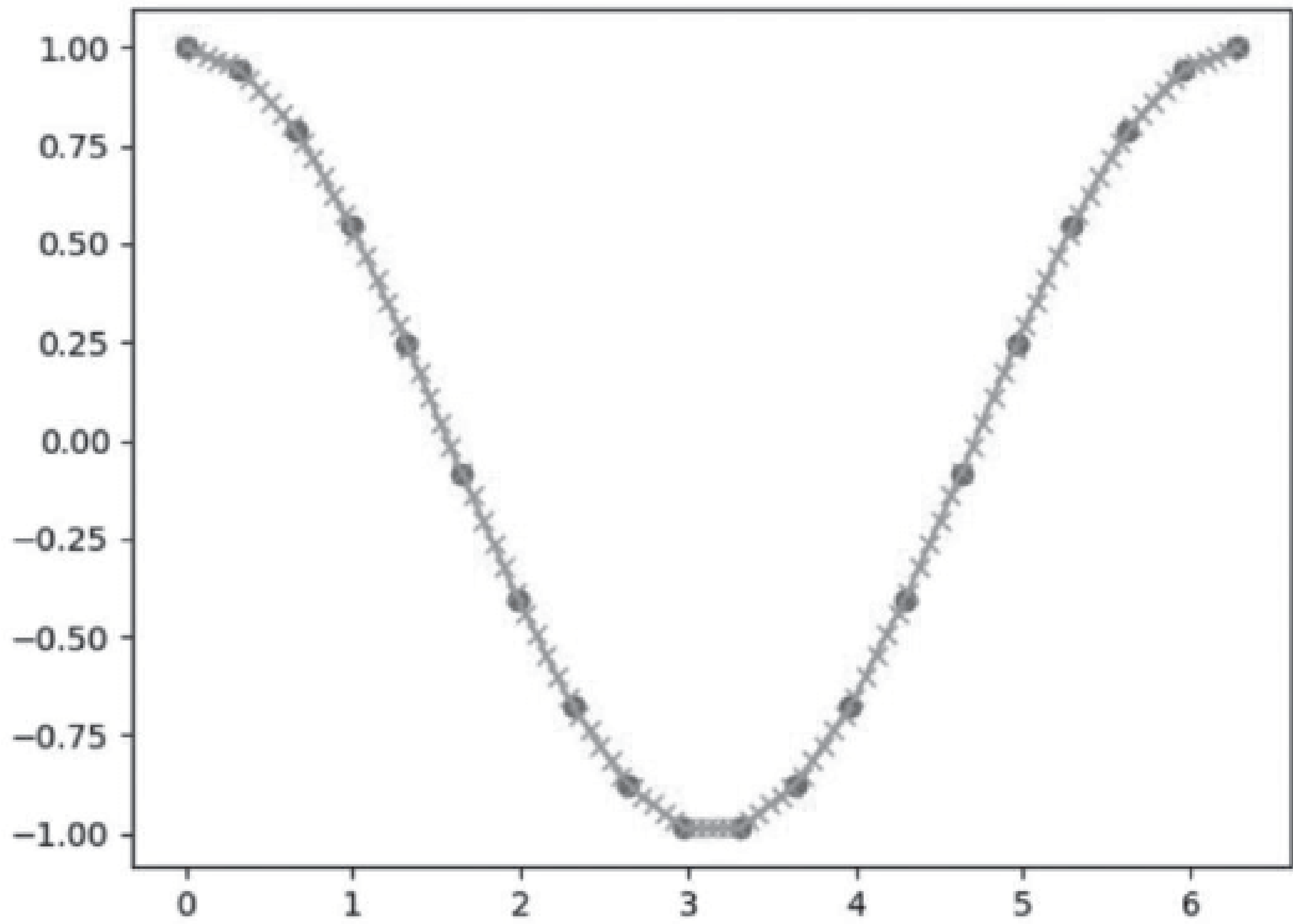
5. 请分别计算下列数组的最大公因子与最小公倍数。（**23-6 节**）

A：[88 108]

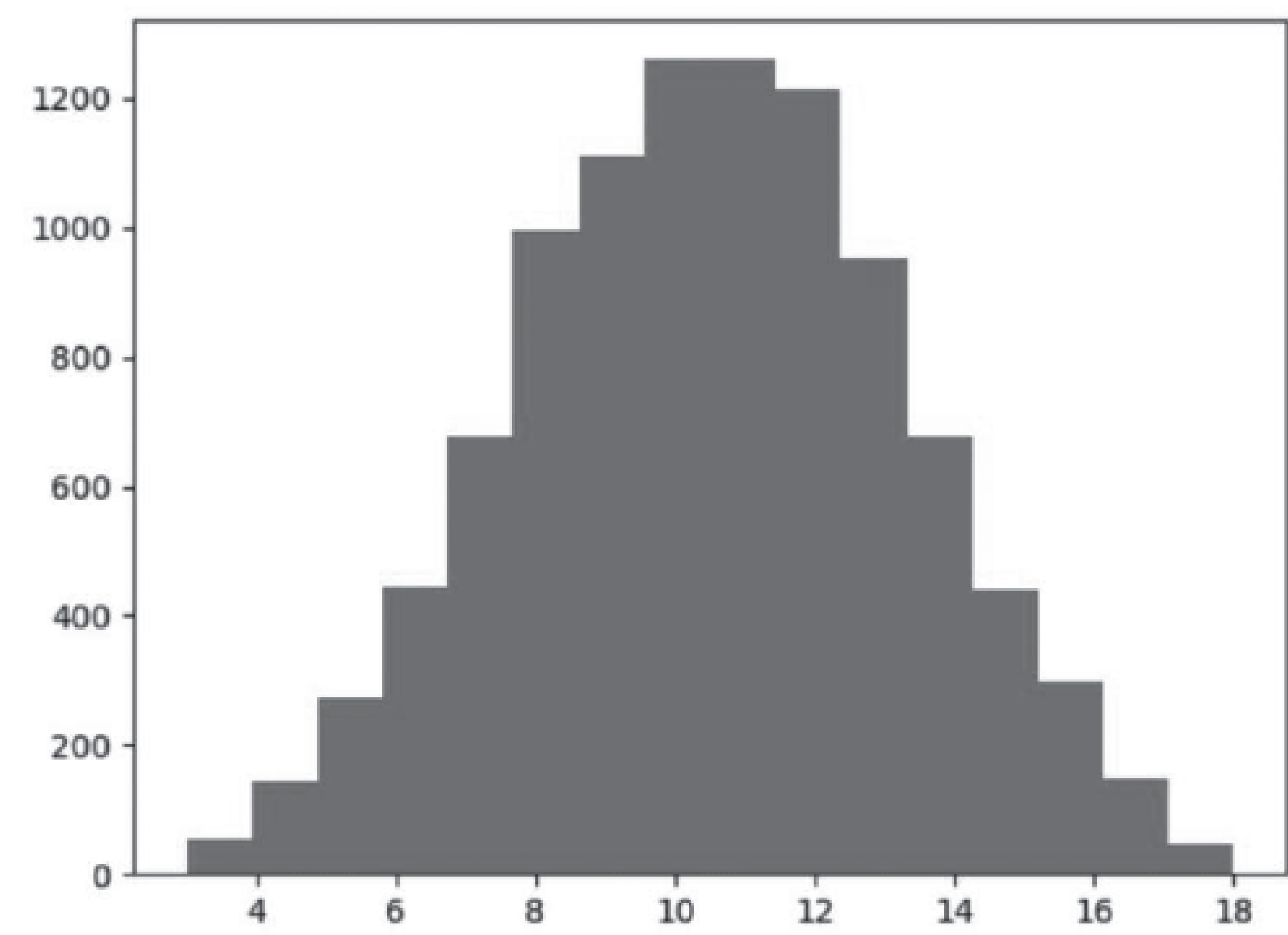
B：[25 35 45 55]

```
===================== RESTART: D:\Python\ex\ex23_5.py =====================
[88 108]最大公因子  4
[25 35 45 55]最大公因子  5
[88 108]最小公倍数  2376
[25 35 45 55]最小公倍数  17325
```

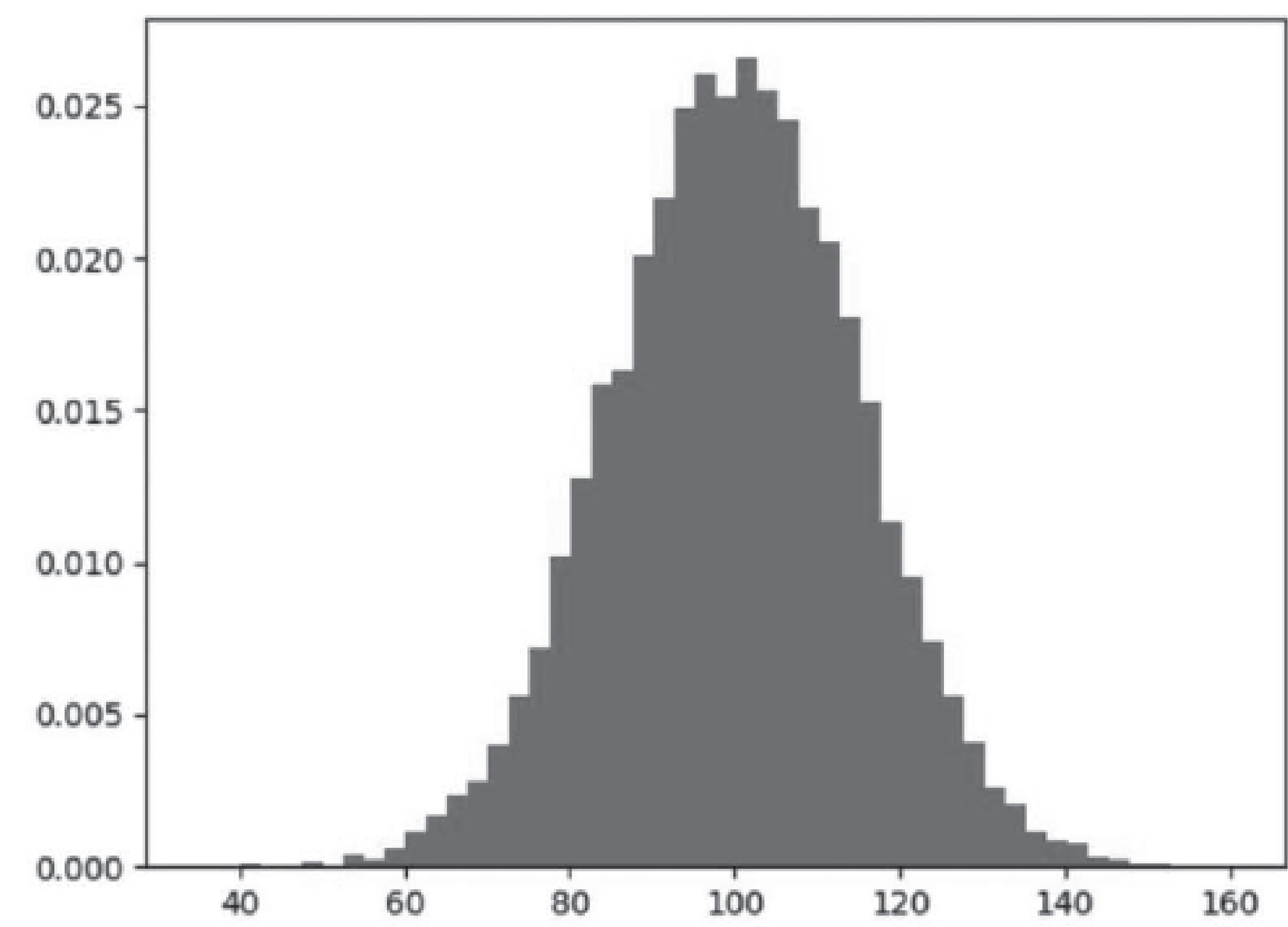
6. 请修改 ch23_4.py 线性插入问题，请将 sin( ) 函数改为 cos( ) 函数，x 轴数值区间是在 0 ～ 2*np.pi 之间。（**23-7 节**）



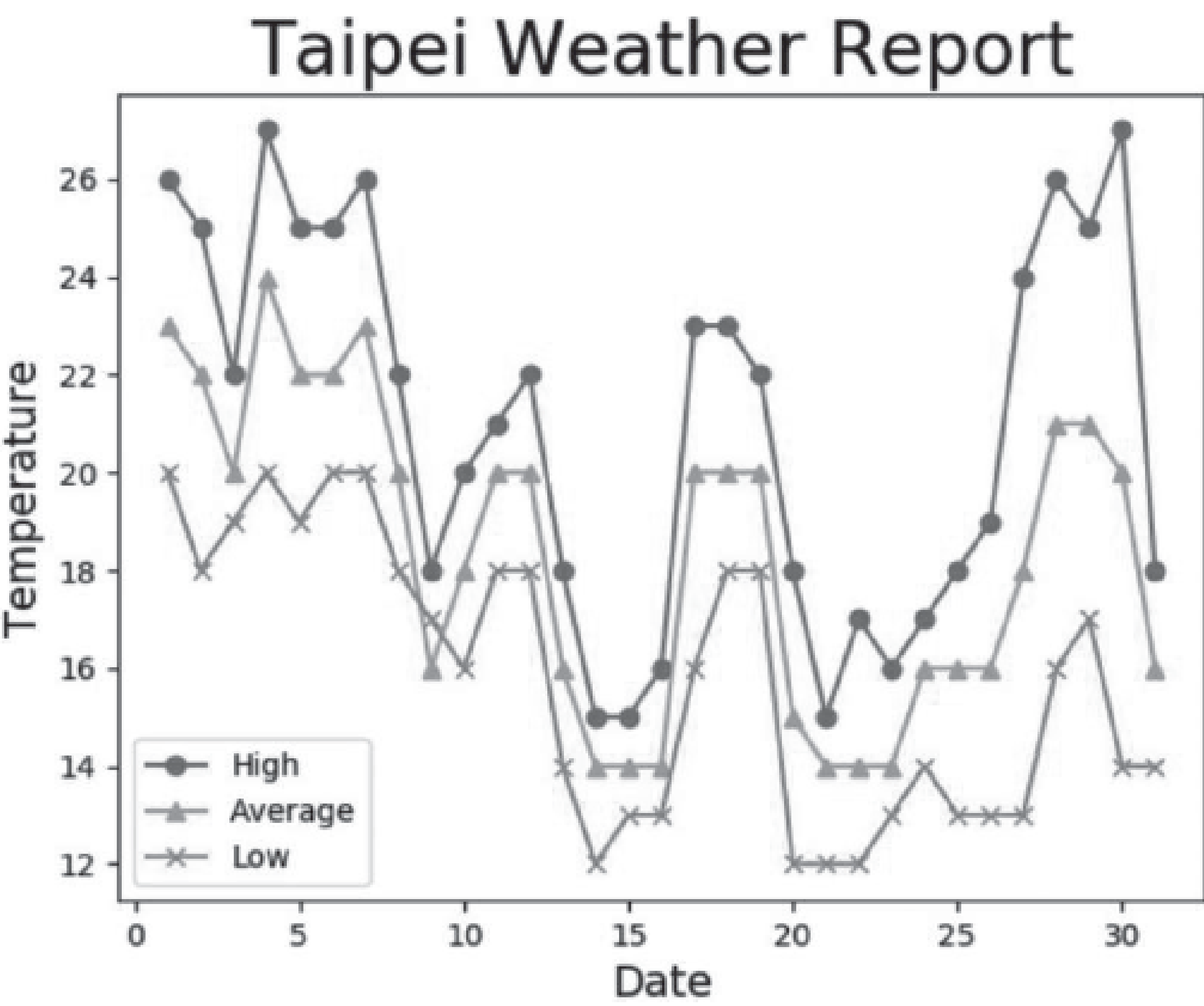
7. 请修改 ch23_5.py，改为 3 颗骰子，同时各掷 10000 次。（**23-8 节**）



8. 请修改 ch23_6.py，将均值 mean 改为 100，将标准偏差 sigma 改为 15，取样改为 10000 次，箱子数 bins 改为 50。（**23-8 节**）



9. 在 ch23 文件夹有一个 weatherTaipei.txt，这个 txt 文件有台北 2020 年 1 月每天最高温度、平均温度与最低温度。请读取此文件建立最高温度、平均温度与最低温度的折线图。（**23-10 节**）



24

第 2 4 章

Scipy 模块

本章摘要

Scipy（可以读成 sigh pie）所代表的是 **Sci**entific **Py**thon，这是一个架构在 Numpy 之上的模块，有了这个模块，可以很顺利地执行统计、优化运算、插值、线性代数、积分、讯号处理、图像处理、常微分方程、快速傅立叶变换等。有一些软件与它的功能类似，例如 Matlab、GNU Octave 和 Scilab。

使用前需安装此模块，方法如下：

```
pip install scipy
```

Scipy 的子模块有许多，本章将介绍最常用的 4 个子模块，读者若想了解更多可以参考下列网址：

https://docs.scipy.org/doc/scipy/reference/

24-1 线性代数 scipy.linalg

更多有关 scipy 内子模块 linalg 的相关知识可以参考下列网址：

https://docs.scipy.org/doc/scipy/reference/linalg.html

在 23-5 节笔者有介绍线性代数运算，当时是使用 numpy.linalg 模块，本节将讲解 scipy.linalg 模块，其实 scipy.linalg 拥有更多进阶的功能与支持。

24-1-1 解联立线性方程式

假设有一个联立方程式如下：

```
3x + 2y = 8
x - y = 1
5y + z = 10
```

如果想要解上述方程式，首先建立 2 个数组，一个是 x、y 和 z 系数的数组，另一个是方程式右边的因变量数组，然后再使用 scipy.linalg 模块的 solve( ) 即可获得 x、y 和 z 的值。

程序实例 ch24_1.py：计算上述联立方程式的值。

```
1  # ch24_1.py
2  import numpy as np
3  from scipy import linalg
4
5  # 定义数组
6  coeff = np.array([[3,2,0],[1,-1,0],[0,5,1]])
7  deps = np.array([8,1,10])
8
9  # 求解
10 ans = linalg.solve(coeff, deps)
11
12 print(ans)
```

执行结果

```
==================== RESTART: D:/Python/ch24/ch24_1.py ====================
[2. 1. 5.]
```

24-1-2　计算行列式 Determinant

行列式（Determinant）的函数式为 det( )，主要是计算正方形矩阵的特别数值，其实这个特性在解联立线性方程式时很有用，同时对于逆矩阵的处理也很有用。通常用 |A| 代表矩阵的行列式。如果是 2×2 的矩阵，行列式的计算方式如下：

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \qquad |A| = ad - bc$$

如果是 3×3 的矩阵，行列式的计算如下：

$$A = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \qquad |A| = a(ei - fh) - b(di - fg) + c(dh - eg)$$

可以将公式想成如下形式：

$$|A| = a \cdot \begin{vmatrix} e & f \\ h & i \end{vmatrix} - b \cdot \begin{vmatrix} d & f \\ g & i \end{vmatrix} + c \cdot \begin{vmatrix} d & e \\ g & h \end{vmatrix}$$

程序实例 ch24_2.py：求 [[1,2],[3,4]] 的行列式。

```
1  # ch24_2.py
2  import numpy as np
3  from scipy import linalg
4
5  A = np.array([[1,2],[3,4]])        # 定义数组
6  x = linalg.det(A)                  # 求解
7  print(x)
```

执行结果

```
==================== RESTART: D:/Python/ch24/ch24_2.py ====================
-2.0
```

24-1-3　特征值和特征向量

特征值（Eigenvalues）和特征向量（Eigenvectors）问题是最常使用的线性代数运算，有一个正方形矩阵 A，可以用下列方式了解特征值（ λ ）和相对应的特征向量（v）。

Av = λv

scipy.linalg 模块内有 eig( ) 可以返回特征值（l，笔者在程序用 l 代替）与特征向量（v），语法如下：

```
l, v = linalg.eig(A)
```

程序实例 ch24_3.py：计算 [[1,2],[3,4]] 的特征值与特征向量。

```
1  # ch24_3.py
2  import numpy as np
3  from scipy import linalg
4
5  A = np.array([[1,2],[3,4]])       # 定义数组
6  l, v = linalg.eig(A)
7  print("特征值   : ", l)
8  print("特征向量 : \n", v)
```

执行结果

```
==================== RESTART: D:\Python\ch24\ch24_3.py ====================
特征值    :  [-0.37228132+0.j  5.37228132+0.j]
特征向量 :
 [[-0.82456484 -0.41597356]
 [ 0.56576746 -0.90937671]]
```

24-2 统计 scipy.stats

更多有关 SciPi 内子模块 stats 的相关知识可以参考下列网址：

https://docs.scipy.org/doc/scipy/reference/stats.html

24-2-1 离散均匀分布 Uniform discrete distribution

在 scipy.stats 模块内有 randint( ) 函数可以建立指定区间均匀分布的随机整数，它的语法如下：

```
stats.randint(low, high, size, options)    # options 是其他不常用的参数
```

上述 low 和 high 是形状变量，实质是最低与最高值，包含 low，但是不包含 high。上述 randint( ) 方法的质量概率函数（probability mass function，pmf）概念如下：

```
pmf(k) = 1/(high - low)                    # for k in low, …,high-1
```

实例 1：建立一个 [0,11) 的概率模型。

```
>>> import scipy.stats as st
>>> rv = st.randint(low=1, high=11)
```

在 scipy.stats 有一个方法是 rvs( ) 可以返回随机数，语法如下：

```
rvs(low, high, loc=0, size=1)              # loc 是均值 mean
```

实例 2：在自建的概率模型 rv 中，产生 6 个随机数。

```
>>> x = rv.rvs(size=6)
>>> print(x)
[5 1 8 2 3 7]
```

其实如果你已经熟悉统计运算，也可以使用下列方式直接产生 6 笔在此模型中的随机数。

实例 3：产生 [0,11) 间的 6 笔随机数。

```
>>> x = st.randint.rvs(low=1, high=11, size=6)
>>> print(x)
[ 5  6  7  4 10  9]
```

在继续说明更多概念前，笔者要介绍几个函数，方便更进一步解说。

质量概率函数（probability mass function）：离散随机数在特定值上的概率，所有特定值的概率总和是 1，参数名称与语法如下：

```
pmf(k, low, high, loc=0)                   # k 是数组
```

实例 4：产生实例 3 的质量概率。

```
>>> rv.pmf(x)
array([0.1, 0.1, 0.1, 0.1, 0.1, 0.1])
```

累积分布函数（cumulative density function）：离散随机数在特定值上的概率累积的值，参数名称与语法如下：

```
cdf(k, low, high, loc=0)              # k 是数组
```

实例 5：产生 [1,2,3,4,5,6,] 的累积分布。

```
>>> rv.cdf([1,2,3,4,5,6])
array([0.1, 0.2, 0.3, 0.4, 0.5, 0.6])
```

百分比函数（percent point function）：返回特定百分比位置的值，相当于是逆 cdf( ) 函数，参数名称与语法如下：

```
ppf(p, low, high, loc=0)              # p 是百分比数组
```

实例 6：延续实例 5，列出百分比位置的值。

```
>>> rv.ppf([0.1, 0.2, 0.3, 0.4, 0.5, 0.6])
array([1., 2., 3., 4., 5., 6.])
```

在 scipy.stats 模块中，对于离散均匀分布的随机数数组模型，可以使用下列统计概念中最常见的函数：

mean(low, high, loc=0)：算术平均数。

var(low, high, loc=0)：变异数。

std(low, high, loc=0)：标准偏差。

median(low, high, loc=0)：中位数。

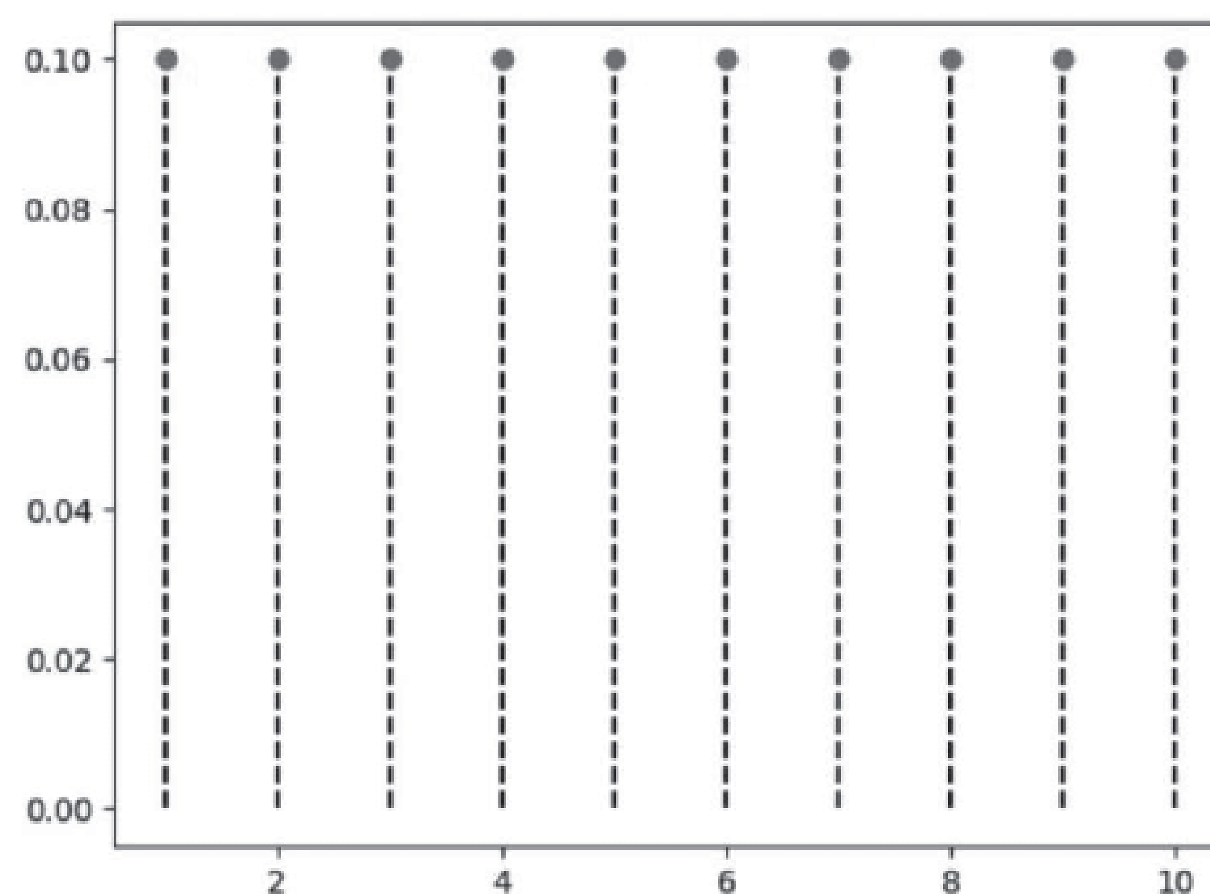
实例 7：延续先前实例，列出 mean( )、var( )、std( )、median( ) 之值。

```
>>> rv.mean()
5.5
>>> rv.var()
8.25
>>> rv.std()
2.8722813232690143
>>> rv.median()
5.0
```

程序实例 ch24_4.py：绘制 [0,11] 间均匀分布的概率模型。

```
1  # ch24_4.py
2  import numpy as np
3  import matplotlib.pyplot as plt
4  import scipy.stats as st
5
6  rv = st.randint(low=1, high=11)
7  x = np.arange(1, 11)
8  plt.plot(x, rv.pmf(x), 'o')
9  plt.vlines(x, 0, rv.pmf(x), linestyles='dashed')
10 plt.show()
```

执行结果



24-2-2　二项分布 Binomial distribution

如果有一个试验，结果只有成功与失败两个结果，同时每次实验均不会受到前一次实验影响，则我们称这是二项分布试验。在这个试验中假设成功概率是 p，则失败概率是 1-p，如果实验次数是 n 次，则成功次数是 np。

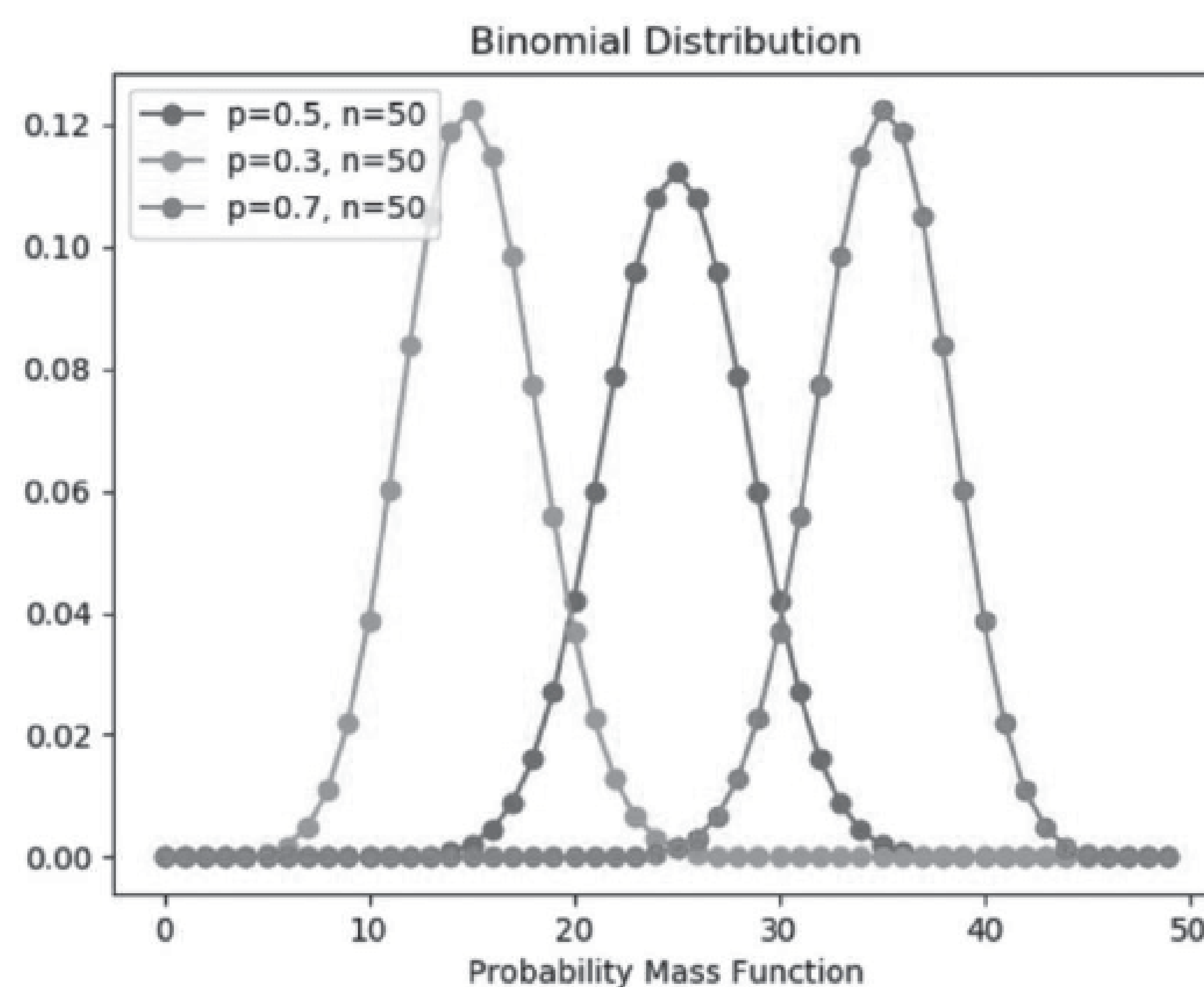
假设实验次数 n_trials 次，实验成功概率是 p，则可以使用下列方式获得二项分布的概率质量与概率累积数组。

```
binom(n_trials, p).pmf(x)          # x 是 0-(n_trials) 之数组
binom(n_trials, p).cdf(x)          # x 是 0-(n_trials) 之数组
```

程序实例 ch24_5.py：绘制 n_trials 是 50 次时，p=0.5, 0.3, 0.7 时二项分布之概率质量函数图。

```
1  # ch24_5.py
2  import numpy as np
3  import matplotlib.pyplot as plt
4  import scipy.stats as st
5
6  n_trials = 50
7  x = np.arange(n_trials)
8
9  plt.plot(x, st.binom(n_trials, 0.5).pmf(x), '-o', label='p=0.5, n=50')
10 plt.plot(x, st.binom(n_trials, 0.3).pmf(x), '-o', label='p=0.3, n=50')
11 plt.plot(x, st.binom(n_trials, 0.7).pmf(x), '-o', label='p=0.7, n=50')
12 plt.title("Binomial Distribution")
13 plt.xlabel("Probability Mass Function")
14 plt.legend()
15 plt.show()
```

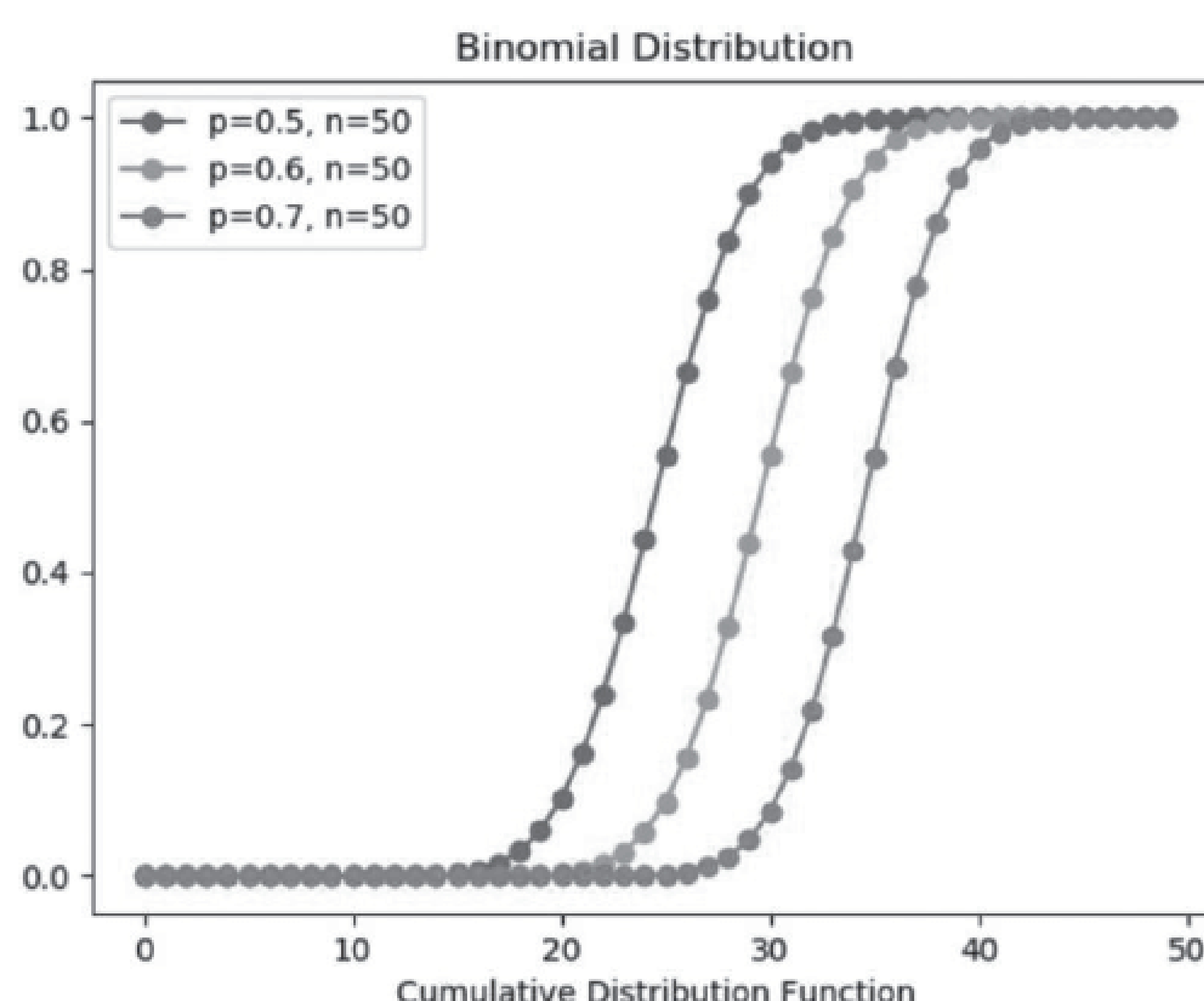
执行结果



程序实例 ch24_6.py：绘制 n_trials 是 50 次时，p=0.5, 0.6, 0.7 时二项分布之概率累积函数图。

```
1  # ch24_6.py
2  import numpy as np
3  import matplotlib.pyplot as plt
4  import scipy.stats as st
5
6  n_trials = 50
7  x = np.arange(n_trials)
8
9  plt.plot(x, st.binom(n_trials, 0.5).cdf(x), '-o', label='p=0.5, n=50')
10 plt.plot(x, st.binom(n_trials, 0.6).cdf(x), '-o', label='p=0.6, n=50')
11 plt.plot(x, st.binom(n_trials, 0.7).cdf(x), '-o', label='p=0.7, n=50')
12 plt.title("Binomial Distribution")
13 plt.xlabel("Cumulative Distribution Function")
14 plt.legend()
15 plt.show()
```

执行结果



24-2-3　连续常态分布

在 scipy 的 stats 统计模块中，可以使用 norm( ) 建立常态分布模型，语法如下：

norm(loc=0, scale=1)　# loc 是 mean 预设是 0，scale 是标准偏差 std 预设是 1

如果 loc 是 0，scale 是 1，上述也可省略，直接使用 norm( )。另外，也可以使用 rvs( )，依据上述模型产生随机数。

实例 1：依据 norm( ) 常态分布模型产生 5 个随机数。

```
>>> import scipy.stats as st
>>> rv = st.norm()
>>> x = rv.rvs(size=5)
>>> print(x)
[-1.35197044 -0.12241552  1.2869465   0.60628621 -0.10141583]
```

有了常态分布模型的随机数数组，就可以使用这些数据建立下列相关的函数值：

概率密度函数（Probability density function）：参数名称与语法如下：

pdf(x, loc=0, scale=1)

实例 2：延续先前实例，建立 5 笔随机数的概率密度函数值。

```
>>> rv.pdf(x)
array([0.15995695, 0.39596426, 0.17428661, 0.33196358, 0.39689595])
```

累积分布函数（Cumulative distribution function）：参数名称与语法如下：

cdf(x, loc=0, scale=1)

实例 3：延续先前实例，建立 5 笔随机数的累积分布函数值。

```
>>> rv.cdf(x)
array([0.08819239, 0.45128497, 0.90094353, 0.72783764, 0.45961018])
```

百分比函数（Percent point function）：参数名称与语法如下：

ppf(x, loc=0, scale=1)

实例 4：产生 [0.5, 0.75] 的百分比值的值。

```
>>> rv.ppf([0.5,0.75])
array([0.        , 0.67448975])
```

在 scipy.stats 模块中，对于连续常态分布的随机数数组模型，可以使用下列统计概念中最常见的函数：

mean(loc=0, scale=1)：算术平均数。

var(loc=0, scale=1)：变异数。

std(loc=0, scale=1)：标准偏差。

median(loc=0, scale=1)：中位数。

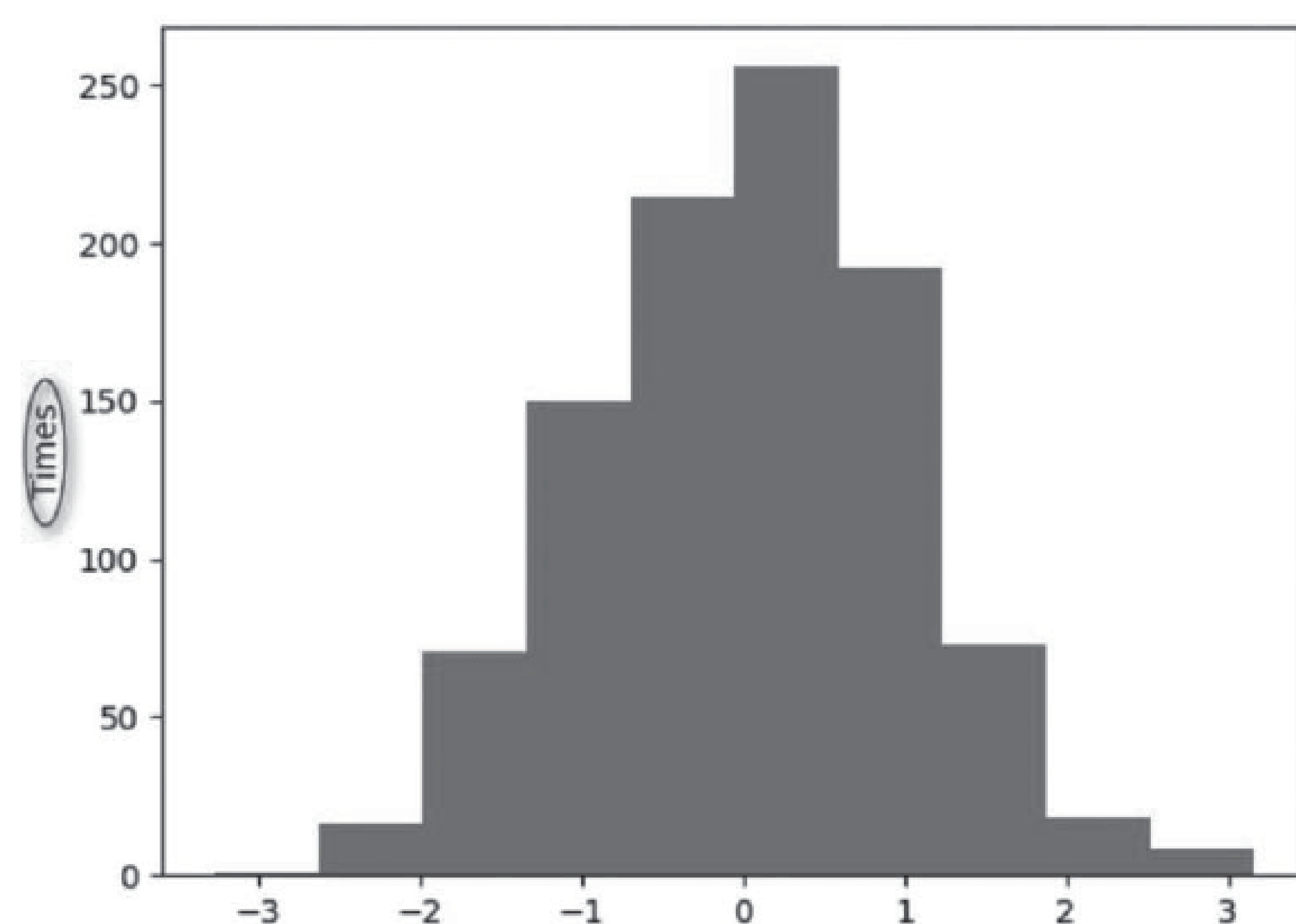
实例 5：延续先前实例，列出 mean( )、var( )、std( )、median( ) 之值。

```
>>> rv.mean()
0.0
>>> rv.var()
1.0
>>> rv.std()
1.0
>>> rv.median()
0.0
```

程序实例 ch24_7.py：使用 norm( ) 产生 1000 个随机数，同时使用直方图 hist( ) 打印结果，请留意 y 轴是纪录次数。

```
1  # ch24_7.py
2  import matplotlib.pyplot as plt
3  import scipy.stats as st
4
5  x = st.norm.rvs(size=1000)
6  plt.hist(x)
7  plt.ylabel("Times")
8  plt.show()
```

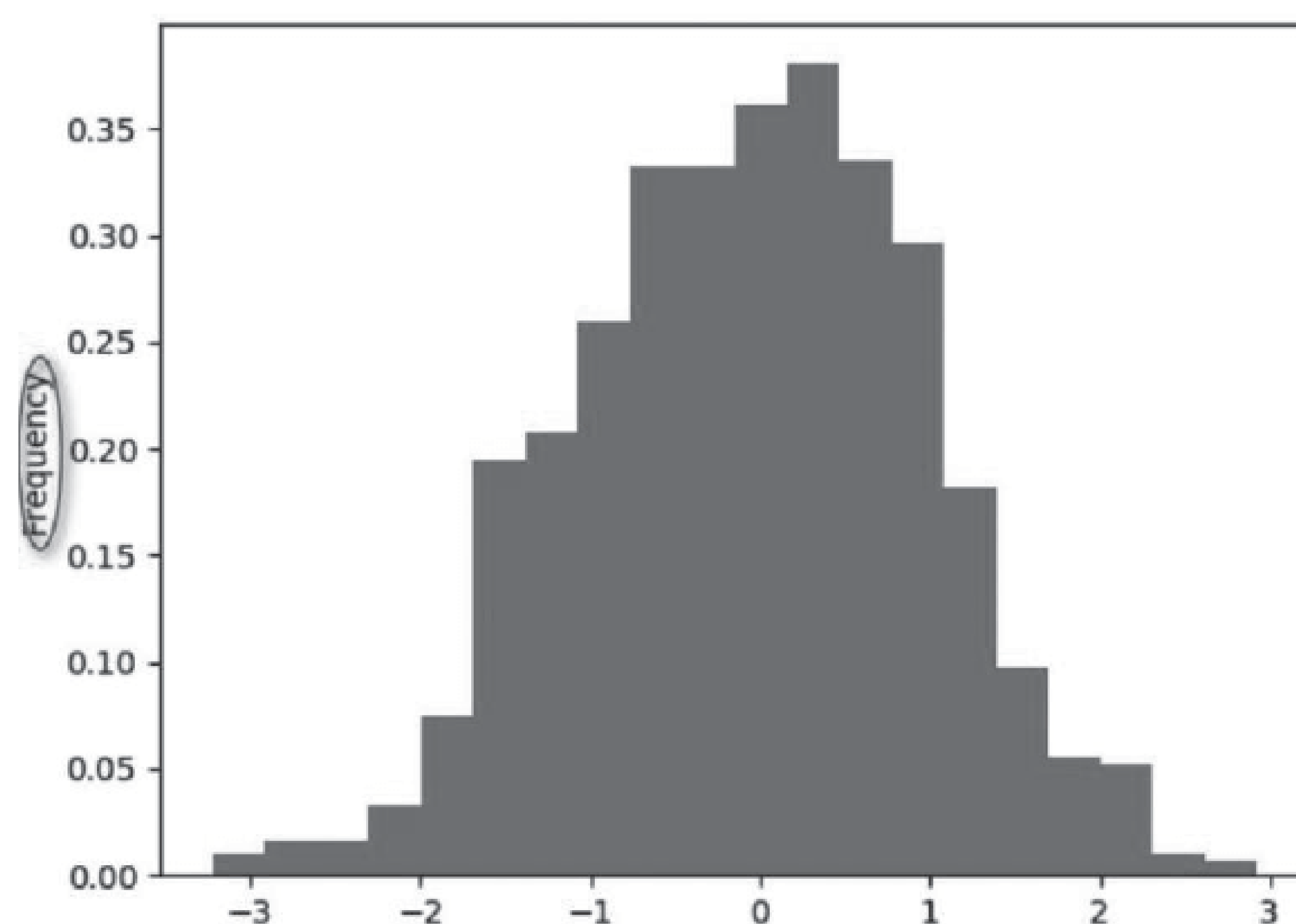
执行结果



程序实例 ch24_8.py：重新设计上一个程序，将 y 轴改为出现频率，同时将 bins 长条数改为 20。

```
1  # ch24_8.py
2  import matplotlib.pyplot as plt
3  import scipy.stats as st
4
5  x = st.norm.rvs(size=1000)
6  plt.hist(x, bins=20, density=True)
7  plt.ylabel("Frequency")
8  plt.show()
```

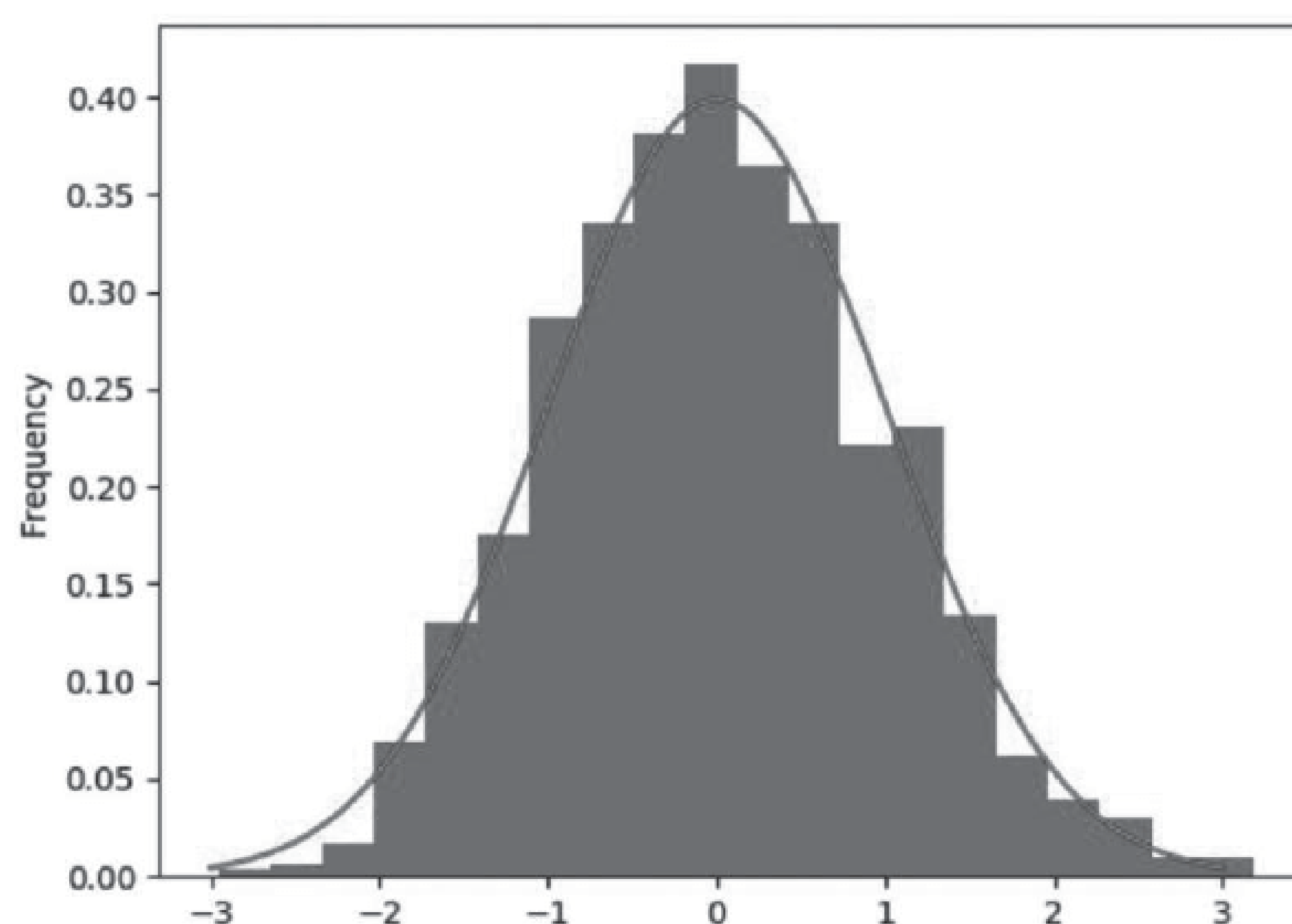
执行结果



程序实例 ch24_9.py：扩充设计 ch24_8.py，以红线绘制概率密度函数产生的值与直方图比较。

```
1  # ch24_9.py
2  import matplotlib.pyplot as plt
3  import scipy.stats as st
4  import numpy as np
5
6  x = st.norm.rvs(size=1000)
7  plt.hist(x, bins=20, density=True)
8  plt.ylabel("Frequency")
9
10 xs = np.linspace(-3,3,100)
11 plt.plot(xs,st.norm.pdf(xs), 'r-')
12
13 plt.show()
```

执行结果



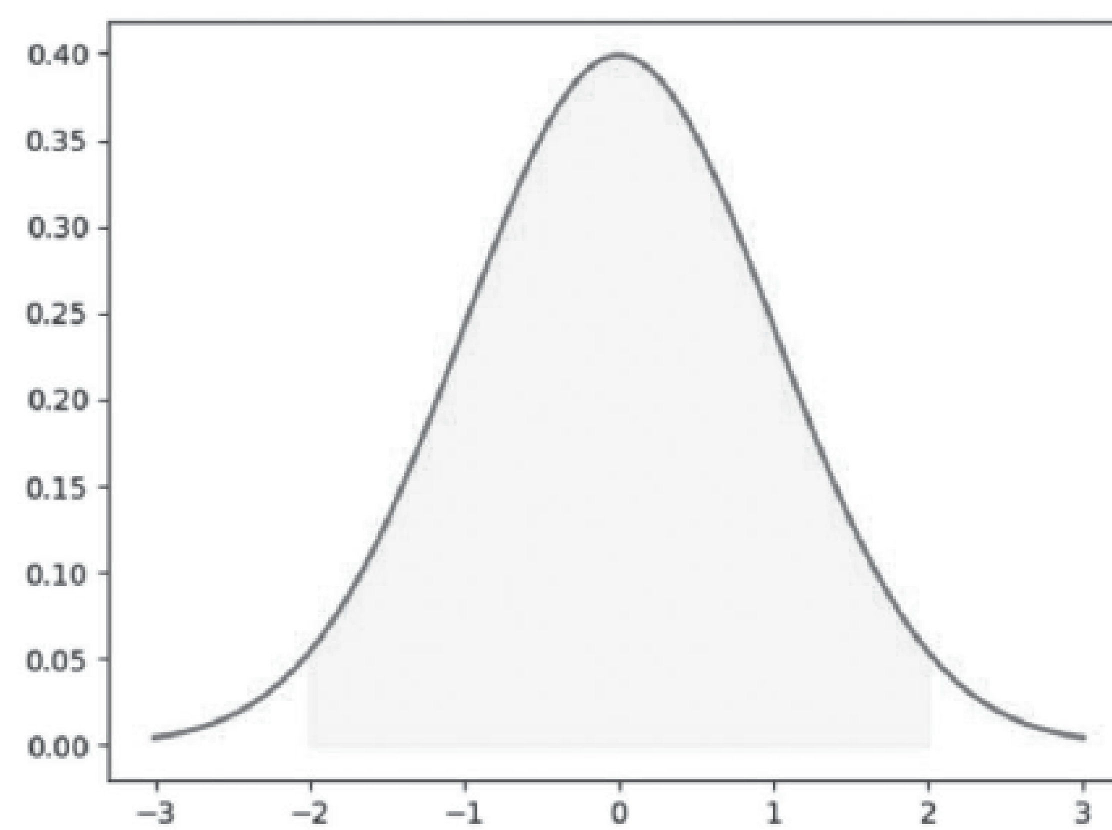
我们也可以使用积分 scipy.integrate 计算落在某个区间的概率值，有关积分的使用可以参考第 4 行和 11 行 trapz( )。

程序实例 ch24_10.py：计算落在 −2 和 2 之间的概率值。

```
1  # ch24_10.py
2  import matplotlib.pyplot as plt
3  import scipy.stats as st
4  from scipy.integrate import trapz
5  import numpy as np
6
7  x = np.linspace(-3,3,100)
8  plt.plot(x, st.norm.pdf(x), 'r-')
9
10 xs = np.linspace(-2,2,100)
11 p = trapz(st.norm.pdf(xs), xs)
12 print("落在-2与2之间的概率是 %4.2f" % (100*p) + "%")
13 plt.fill_between(xs, st.norm.pdf(xs), color="yellow")
14
15 plt.show()
```

执行结果

```
==================== RESTART: D:\Python\ch24\ch24_10.py ====================
落在-2与2之间的概率是 95.45%
```

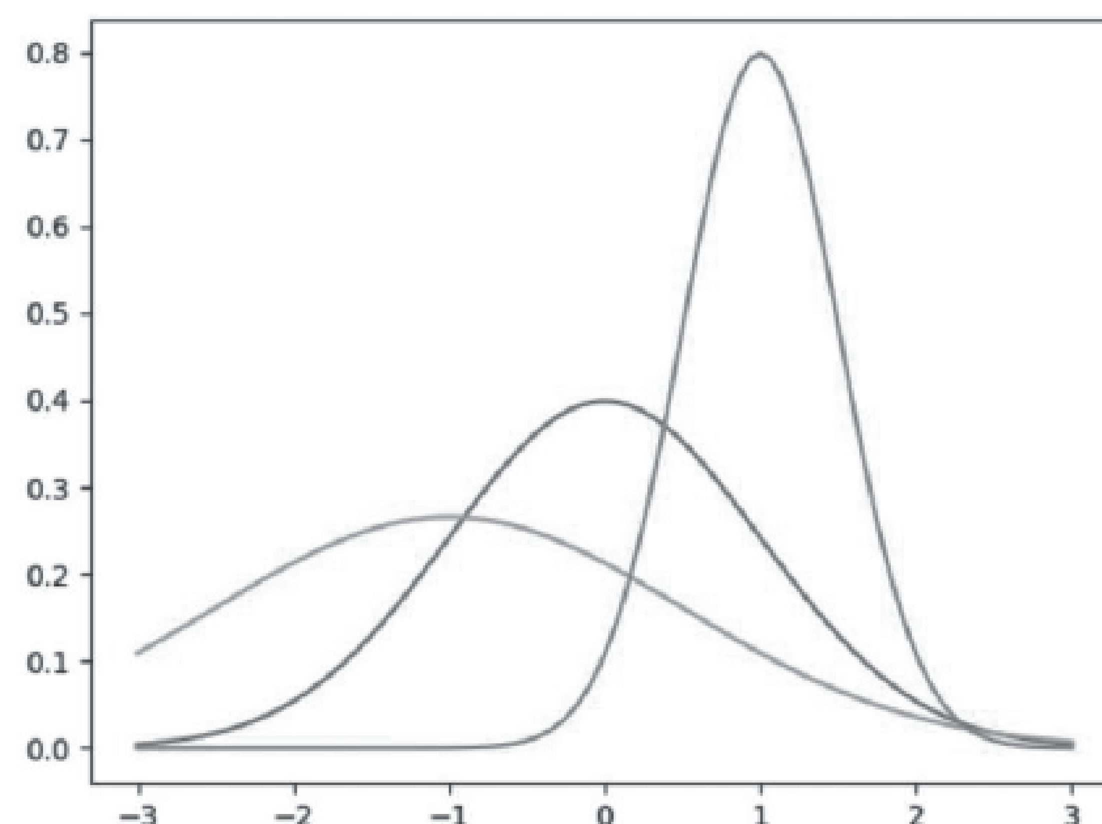


其实上述是以均值 loc 是 1，标准偏差 scale 是 1 的情况解说，适度更改 loc 和 scale，将看到不同的连续常态分布曲线。

程序实例 ch24_11.py：loc 和 scale 分别是 0,1、−1,2、1,0.5，绘制连续常态分布曲线。

```
1  # ch24_11.py
2  import matplotlib.pyplot as plt
3  import scipy.stats as st
4  import numpy as np
5
6  x = np.linspace(-3,3,100)
7  plt.plot(x, st.norm.pdf(x, loc=0, scale=1))
8  plt.plot(x, st.norm.pdf(x, loc=-1, scale=1.5))
9  plt.plot(x, st.norm.pdf(x, loc=1, scale=0.5))
10 plt.show()
```

执行结果



24-3 优化 scipy.optimize

更多有关 SciPi 内子模块 optimize 的相关知识可以参考下列网址：

https://docs.scipy.org/doc/scipy/reference/optimize.html#module-scipy.optimize

optimize 模块内有许多功能，如处理优化、找最小值、曲线拟合、解方程式的根等。其实这些概念需有线性代数（Linear Algebra）和优化（Optimization）基础，在此笔者将简单介绍解方程式方面的问题。

24-3-1　解一元二次方程式的根

在 5-7-4 节笔者有介绍使用 Python 基本功解方程式的根，其实我们也可以使用 optimization.root( ) 解方程式的根，它的语法如下：

```
root(fun, x0, options, …)            # options 是较少用的参数
```

fun 是要解的函数名称，x0 是初始迭代值（可以用不同的参数值，会有不同的结果）。

程序实例 ch24_12.py：计算下列一元二次方程式的根。

$3x^2 + 5x + 1 = 0$

```
1  # ch24_12.py
2  from scipy.optimize import root
3  def f(x):
4      return (a*x**2 + b*x + c)
5
6  a = 3
7  b = 5
8  c = 1
9  r1 = root(f,0)              # 初始迭代值0
10 print(r1.x)
11 r2 = root(f,-1)             # 初始迭代值-1
12 print(r2.x)
```

执行结果

```
==================== RESTART: D:/Python/ch24/ch24_12.py ====================
[-0.23240812]
[-1.43425855]
```

24-3-2　解联立线性方程式

我们也可以使用 root( ) 方法解联立方程式问题，可以参考下列实例。

程序实例 ch24_13.py：计算下列联立线性方程式的值。

```
2x + 3y = 13              # 相当于 2x + 3y - 13 = 0
x - 2y = -4               # 相当于 x - 2y + 4 = 0
```

在套用 root( ) 方法中，x 相当于 x[0]，y 相当于 x[1]。

```
1  # ch24_13.py
2  from scipy.optimize import root
3  def fun(x):
4      return (a*x[0]+b*x[1]+c, d*x[0]+e*x[1]+f)
5
6  a = 2
7  b = 3
8  c = -13
9  d = 1
10 e = -2
11 f = 4
12 r =  root(fun,[0,0])     # 初始迭代值0, 0
13 print(r.x)
```

执行结果

```
==================== RESTART: D:/Python/ch24/ch24_13.py ====================
[2. 3.]
```

24-3-3 计算 2 个线性方程式的交叉点

root( ) 方法也可以寻找 2 个线性方程式的交叉点。

程序实例 24_14.py：例如有 2 个线性方程式如下，请找出交叉点。

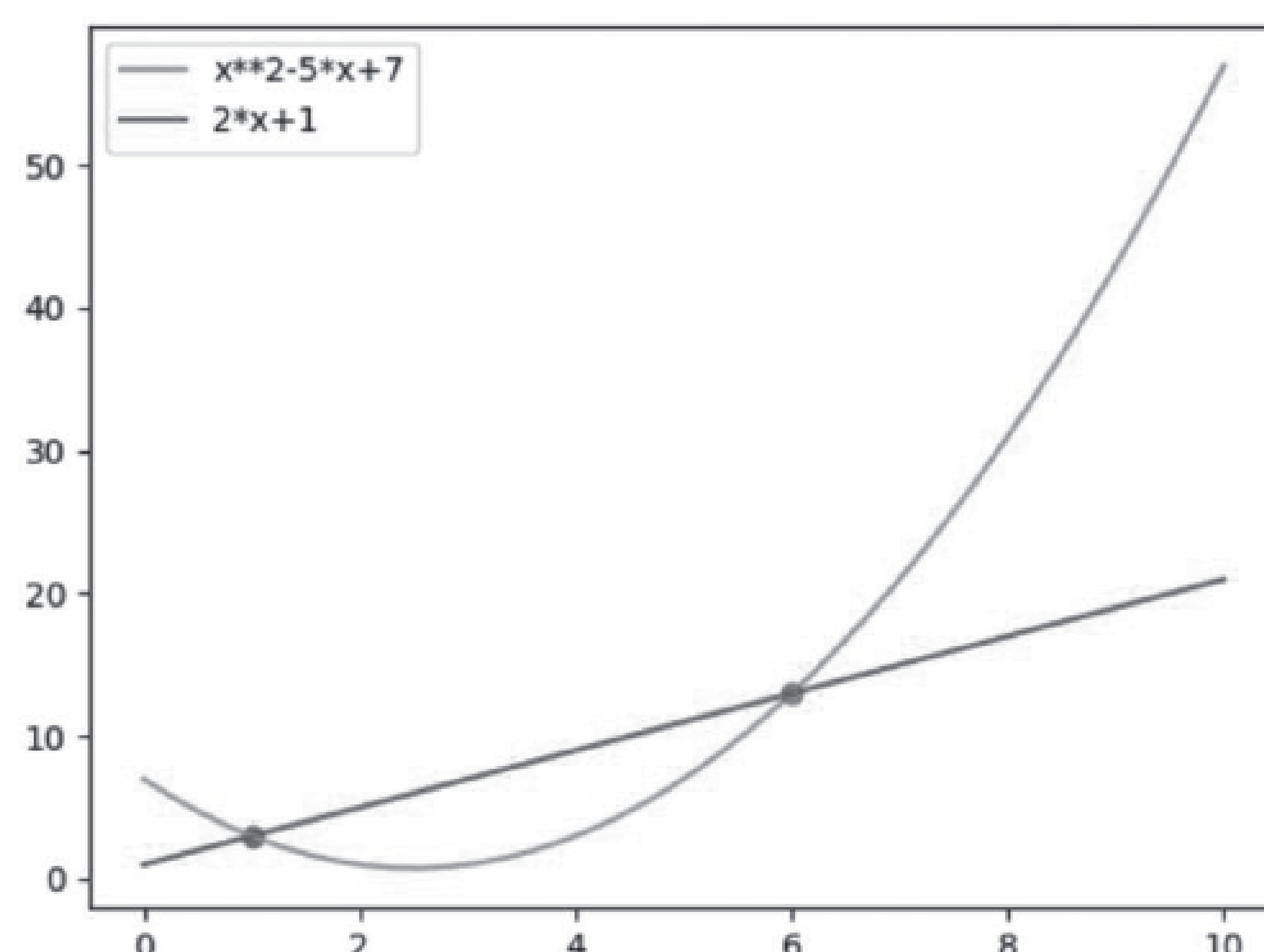
$$f(x) = x^2 - 5x + 7$$

$$f(x) = 2x + 1$$

```
1  # ch24_14.py
2  from scipy.optimize import root
3  import matplotlib.pyplot as plt
4  import numpy as np
5  def fx(x):
6      return (x**2-5*x+7)
7
8  def fy(x):
9      return (2*x+1)
10
11 # 计算交叉点
12 r1 =  root(lambda x:fx(x)-fy(x), 0)      # 初始迭代值0
13 r2 =  root(lambda x:fx(x)-fy(x), 5)      # 初始迭代值5
14 print("x1 = %4.2f,  y1 = %4.2f" % (r1.x,fx(r1.x)))
15 print("x2 = %4.2f,  y2 = %4.2f" % (r2.x,fx(r2.x)))
16 # 绘制fx函数图形
17 x1 = np.linspace(0, 10, 40)
18 y1 = x1**2-5*x1+7                        # fx
19 plt.plot(r1.x, fx(r1.x), 'o')
20 plt.plot(x1, y1, '-', label='x**2-5*x+7')
21 # 绘制fy函数图形
22 x2 = np.linspace(0, 10, 40)
23 y2 = 2*x2+1                              # fy
24 plt.plot(r2.x, fy(r2.x), 'o')
25 plt.plot(x2, y2, '-', label='2*x+1')
26 plt.legend(loc='best')
27 plt.show()
```

执行结果

```
==================== RESTART: D:\Python\ch24\ch24_14.py ====================
x1 = 1.00,  y1 = 3.00
x2 = 6.00,  y2 = 13.00
```



24-3-4　找出线性方程式的最小值和最大值

一元二次方程式表达如下：

f(x) = ax^2 + bx + c

如果 a > 0，代表函数曲线开口向上，所以可以找到此线性函数 f(x) 的最小值。如果 a < 0，代表函数曲线开口向下，所以可以找到此线性函数 f(x) 的最大值。

在 optimize 模块内有 minimize_scalar( ) 方法可以找出 f(x) 函数的最小值，也可以由此导入函数找出最小值的（x,y）坐标，语法如下：

minimize_scalar(fun)

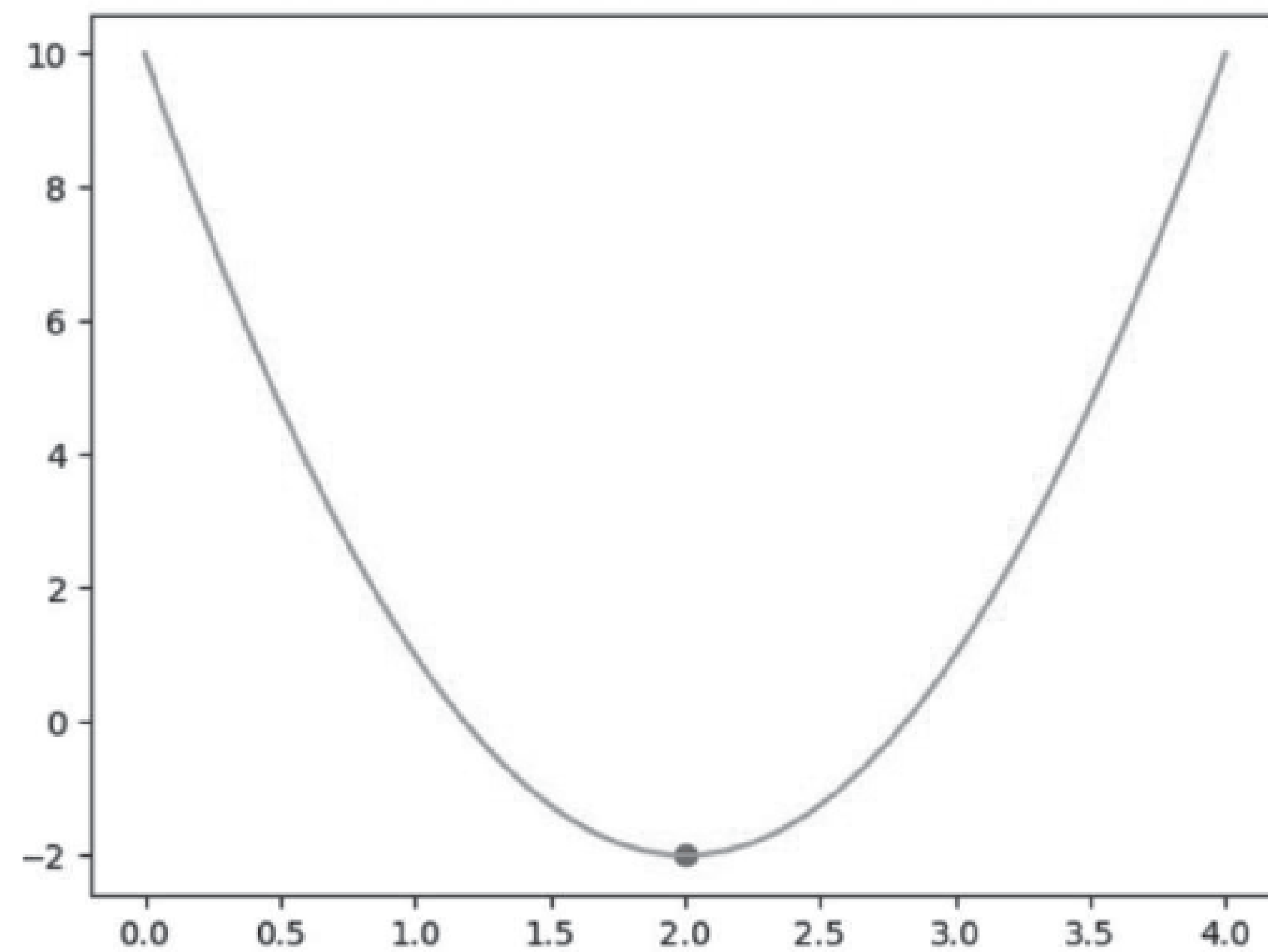
程序实例 ch24_15.py：找出下列函数的最小值与其坐标，同时绘制此函数图形。

f(x) = $3(x-2)^2$ - 2

```
1  # ch24_15.py
2  from scipy.optimize import root
3  from scipy.optimize import minimize_scalar
4  import matplotlib.pyplot as plt
5  import numpy as np
6  def f(x):
7      return (3*(x-2)**2 - 2)
8
9  # 计算最小值
10 r = minimize_scalar(f)
11 print("当x是 %4.2f 时, 有函数最小值" % r.x)
12 print("坐标是 ", r.x, f(r.x))
13 # 绘制此函数图形
14 x = np.linspace(0, 4, 40)
15 y = 3*(x-2)**2 - 2
16 plt.plot(r.x, f(r.x), 'o')
17 plt.plot(x, y, '-')
18 plt.show()
```

执行结果

```
===================== RESTART: D:\Python\ch24\ch24_15.py =====================
当x是 2.00 时, 有函数最小值
坐标是  2.0 -2.0
```



使用 minimize_scalar( ) 是可以找出 f(x) 函数最小值的方法，只要此 f(x) 在返回时乘以 −1，即可找出 f(x) 函数的最大值。

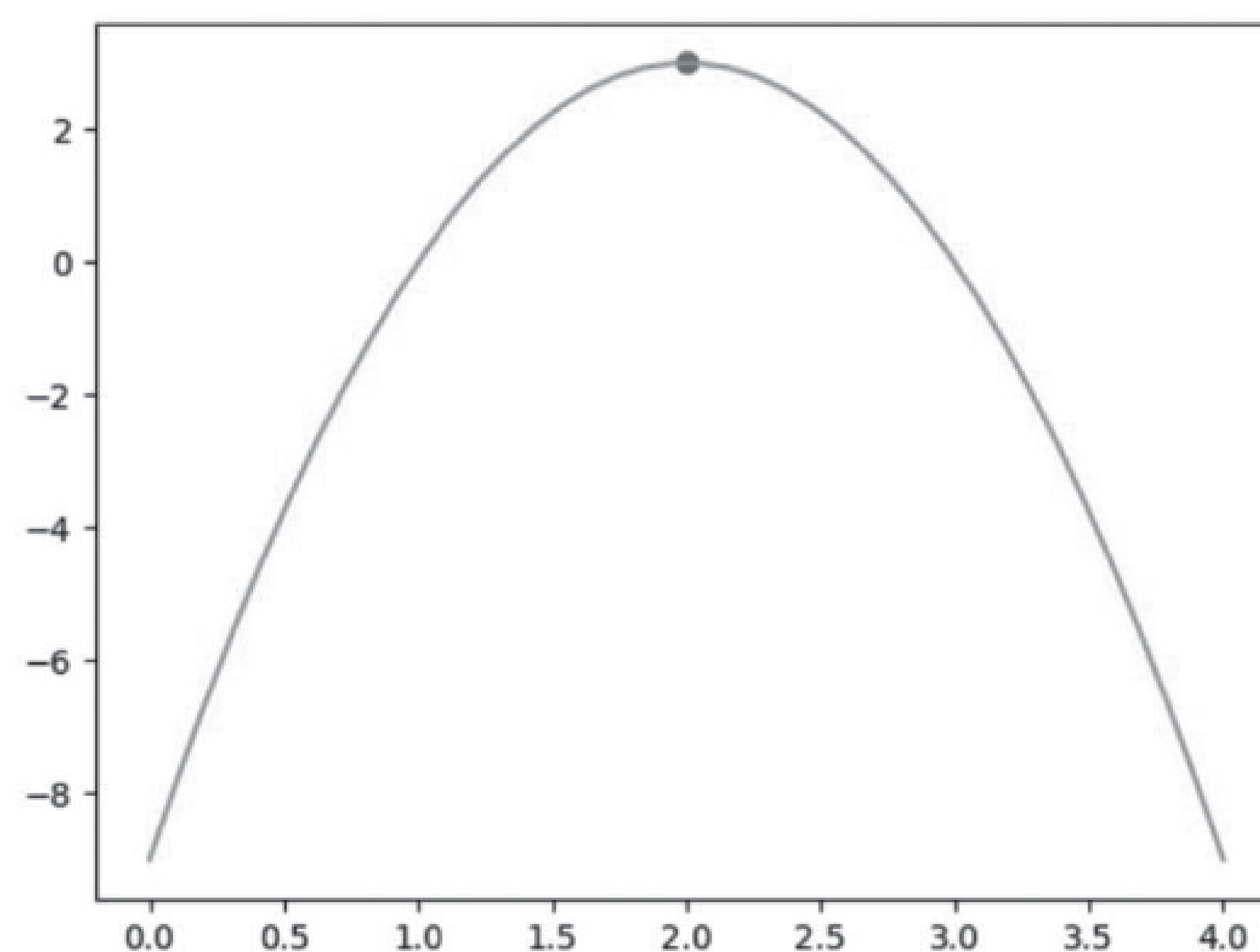
程序实例 ch24_16.py：找出下列函数的最大值与其坐标。

$$f(x) = -3(x-2)^2 + 3$$

```
1  # ch24_16.py
2  from scipy.optimize import root
3  from scipy.optimize import minimize_scalar
4  import matplotlib.pyplot as plt
5  import numpy as np
6  def fmax(x):
7      return (-1*(-3*(x-2)**2 + 3))
8
9  def f(x):
10     return (-3*(x-2)**2 + 3)
11
12 # 计算最大值
13 r = minimize_scalar(fmax)
14 print("当x是 %4.2f 时，有函数最大值" % r.x)
15 print("坐标是 ", r.x, f(r.x))
16 # 绘制此函数图形
17 x = np.linspace(0, 4, 40)
18 y = -3*(x-2)**2 + 3
19 plt.plot(r.x, f(r.x), 'o')
20 plt.plot(x, y, '-')
21 plt.show()
```

执行结果

```
===================== RESTART: D:\Python\ch24\ch24_16.py =====================
当x是 2.00 时，有函数最大值
坐标是  2.0 3.0
```



24-4 插值 scipy.interpolate

更多有关 SciPi 内子模块 interpolate 的相关知识可以参考下列网址：

https://docs.scipy.org/doc/scipy/reference/tutorial/interpolate.html

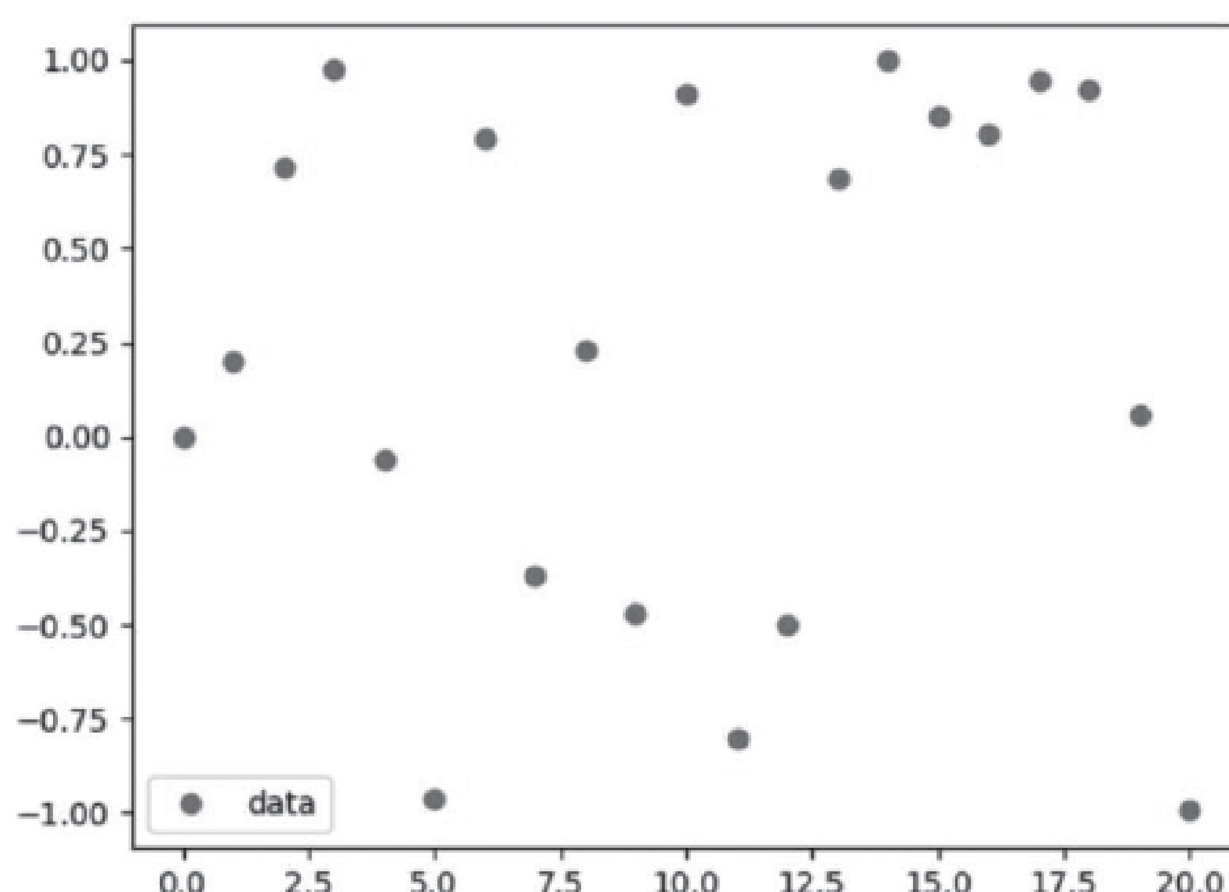
本节将只介绍差值中最简单的 1-D 插值，更多的应用读者可以参考相关书籍。在科学运算领域 interpolate 可以翻译为**插值**或**内插**，本书翻译为**插值**，它的概念是在一些已知的数据（可以从实验或采样取得，如已知的离散的点）中，使用插入方法推算新的点。

程序实例 ch24_17.py：有一些采样所得的散点共计 21 个，这些散点可用函数 f(x) 公式表示，此例中在 0 和 20 之间产生 21 个点，本程序将绘出这些点。

$$f(x) = \sin(x^2/5)$$

```
1  # ch24_17.py
2  import numpy as np
3  import matplotlib.pyplot as plt
4  from scipy.interpolate import interp1d
5
6  x = np.linspace(0,20,21)
7  y = np.sin(x**2/5.0)
8  plt.plot(x,y,'o',label='data')
9  plt.legend(loc='best')
10 plt.show()
```

执行结果



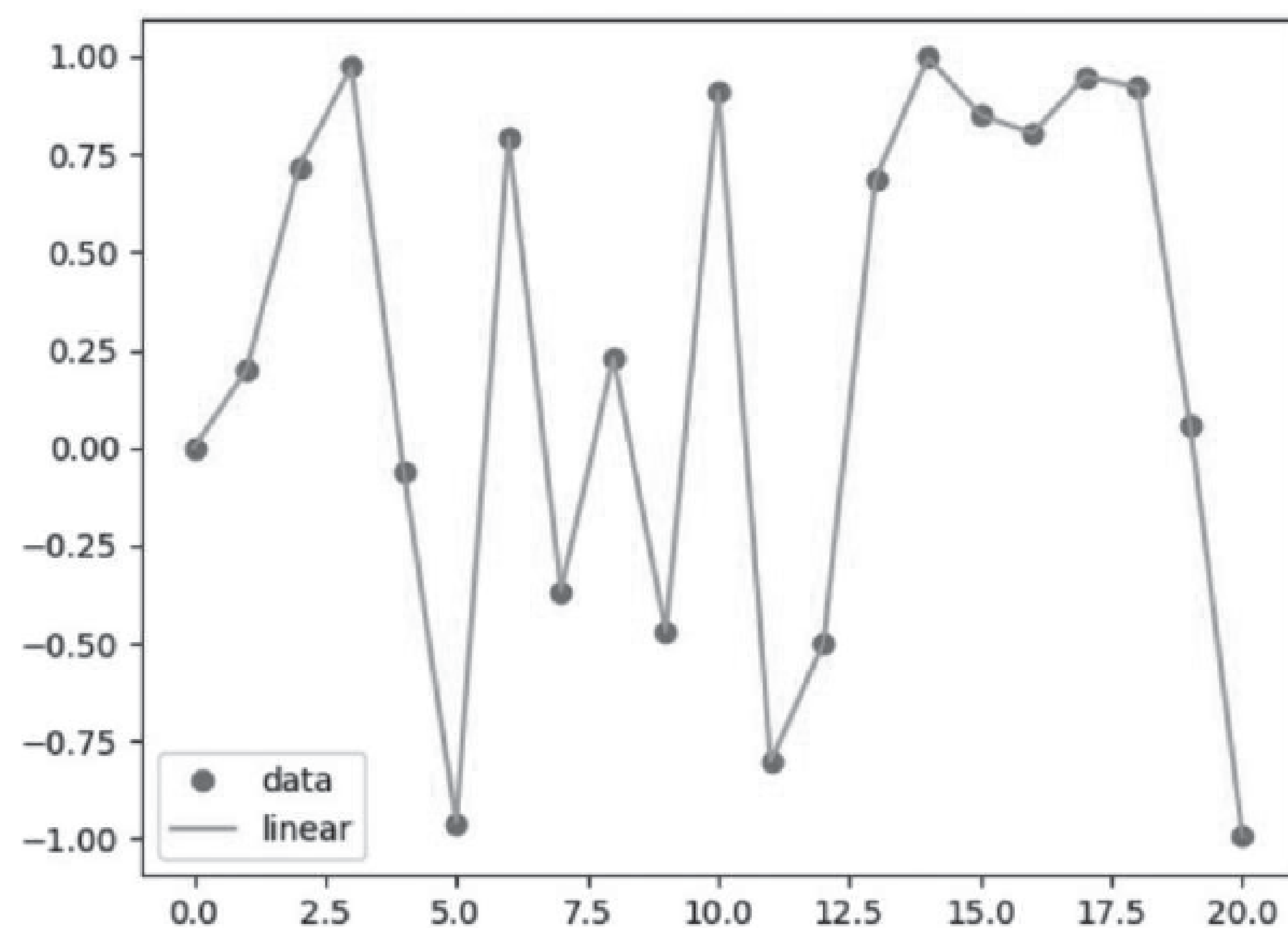
现在我们可以使用下列 Linear 方法（线性插值法），将上述（x,y）数据导入 interp1d( )，产生新的函数 fLinear( )。

```
flinear = interp1d(x, y)                # 预设是 linear 方法
```

程序实例 ch24_18.py：使用 Linear 插入方法扩充程序实例 ch24_17.py，同时将 x 的点扩充至 61 个。

```
1  # ch24_18.py
2  import numpy as np
3  import matplotlib.pyplot as plt
4  from scipy.interpolate import interp1d
5
6  x = np.linspace(0,20,21)
7  y = np.sin(x**2/5.0)
8
9  fLinear = interp1d(x,y)                               # Linear插值函数
10 xnew = np.linspace(0,20,61)                           # 扩充的x轴数据
11
12 plt.plot(x,y,'o',label='data')
13 plt.plot(xnew,fLinear(xnew),'-',label='linear')       # Linear
14
15 plt.legend(loc='best')
16 plt.show()
```

执行结果



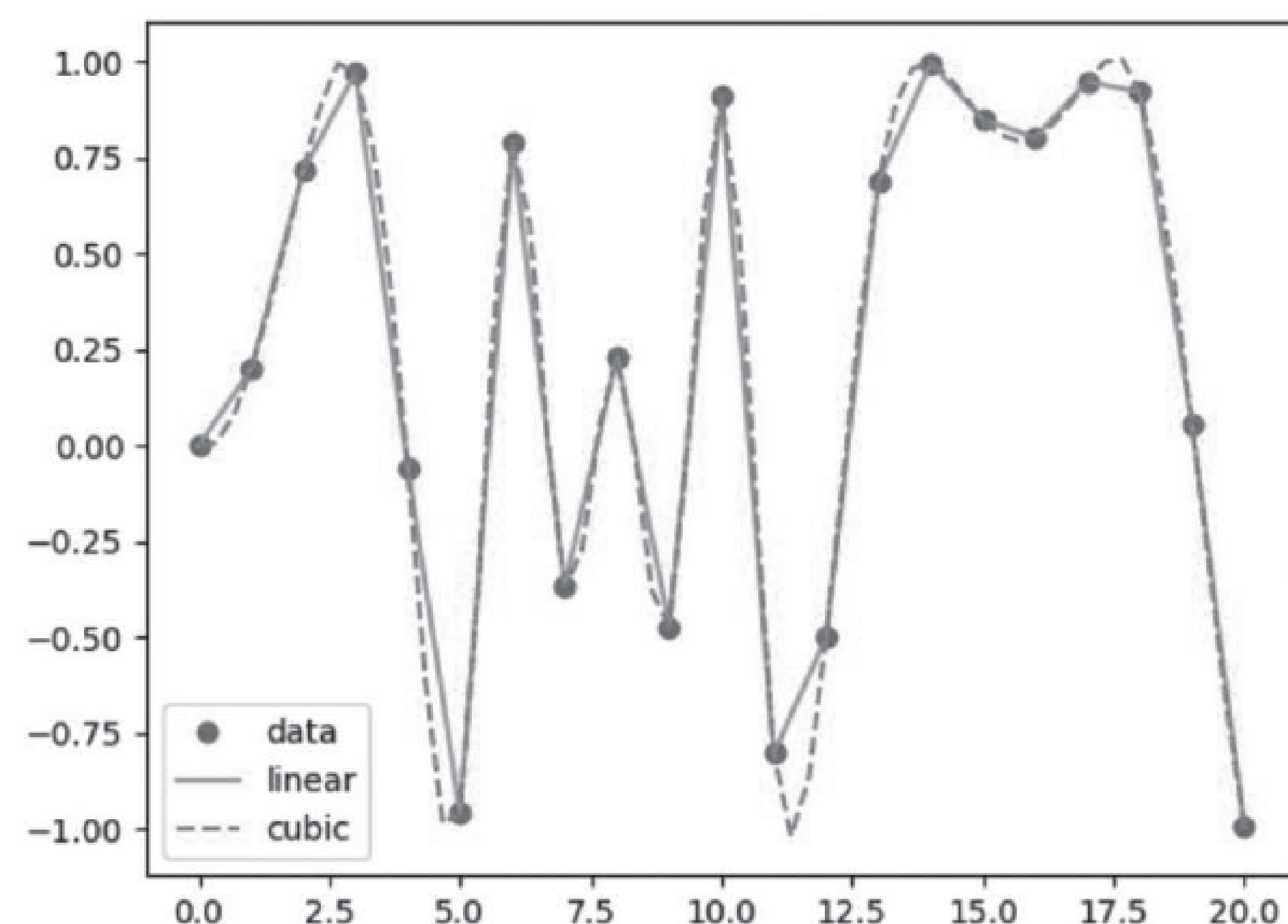
现在我们可以使用下列 Cubic 方法（三次插值法），将上述（x,y）数据导入 interp1d( )，产生新的 fCubic 对象。

```
fCubic = interp1d(x, y, kind='cubic')
```

程序实例 ch24_19.py：使用 Cubic 插入方法扩充程序实例 ch24_18.py。

```
1  # ch24_19.py
2  import numpy as np
3  import matplotlib.pyplot as plt
4  from scipy.interpolate import interp1d
5
6  x = np.linspace(0,20,21)
7  y = np.sin(x**2/5.0)
8
9  fLinear = interp1d(x,y)                          # Linear插值函数
10 fCubic = interp1d(x,y,kind='cubic')              # Cubic插值函数
11 xnew = np.linspace(0,20,61)                      # 扩充的x轴数据
12
13 plt.plot(x,y,'o',label='data')
14 plt.plot(xnew,fLinear(xnew),'-',label='linear')  # Linear
15 plt.plot(xnew,fCubic(xnew),'--',label='cubic')   # Cubic
16
17 plt.legend(loc='best')
18 plt.show()
```

执行结果



习题

1. 请使用 linalg 模块解下列联立方程式。（24-1 节）

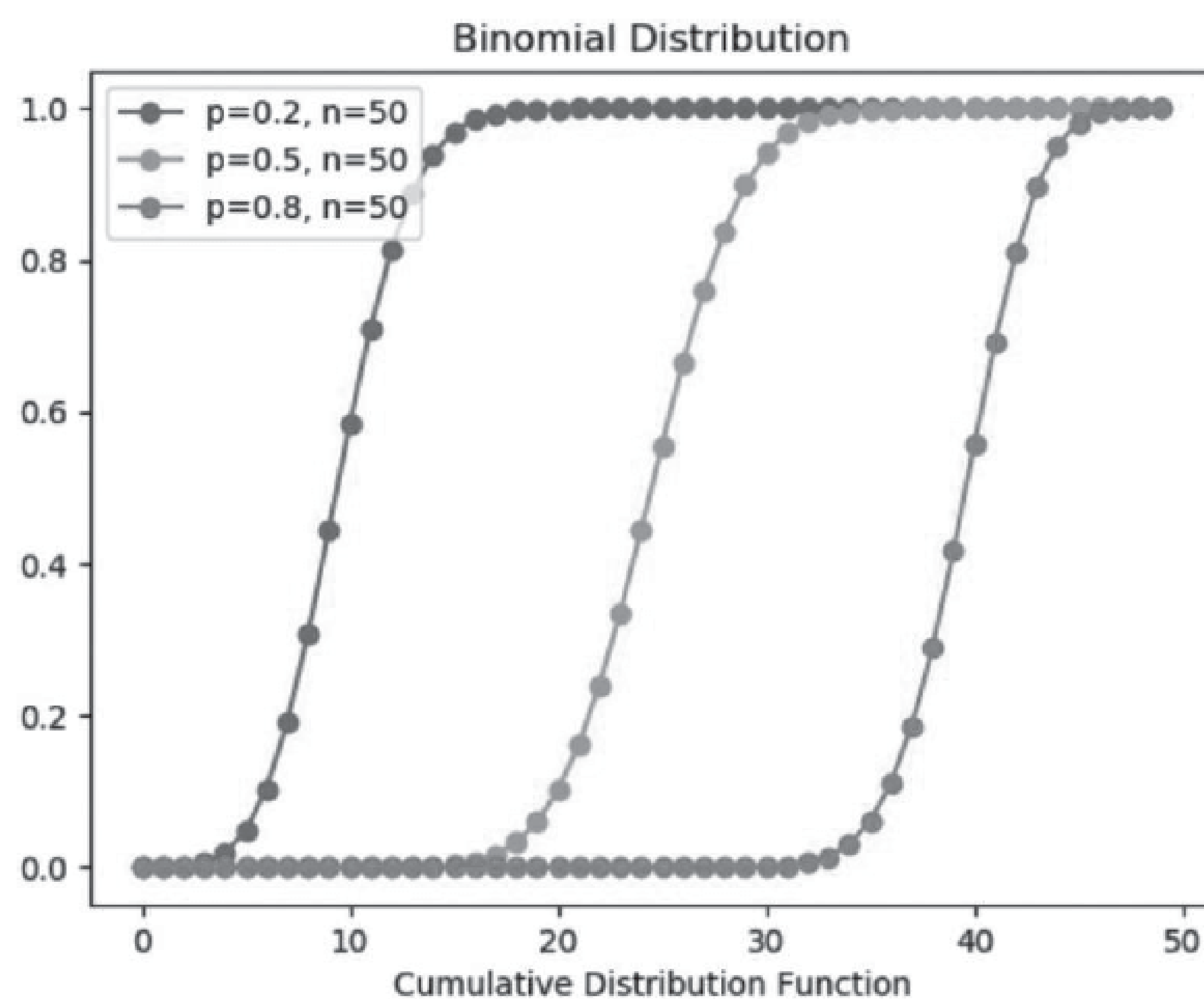
x + 3y + 5z = 20

2x + 5y + z = 12

2x + 3y + 8z = 6

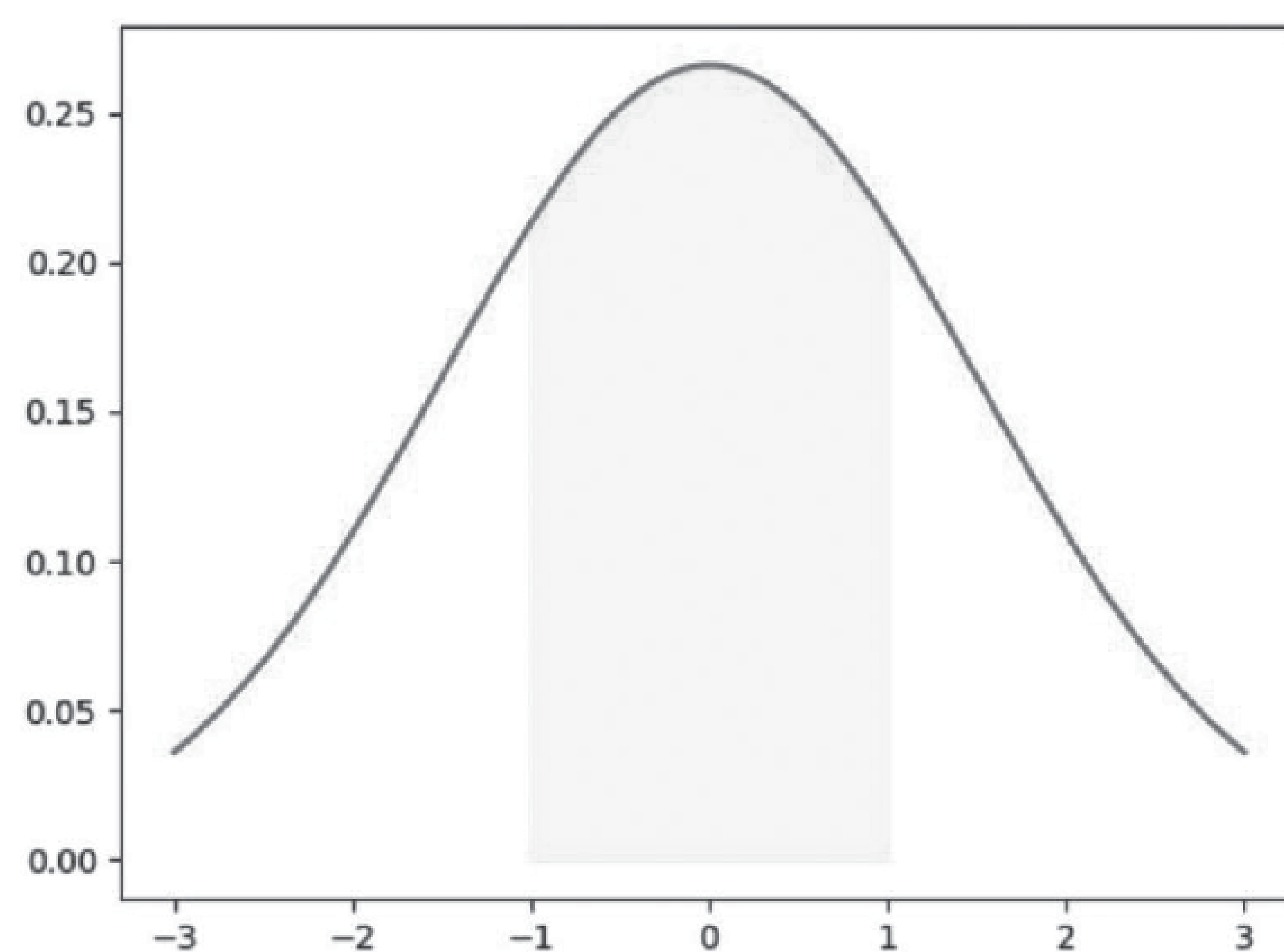
```
====================== RESTART: D:/Python/ex/ex24_1.py ======================
[-20.  10.   2.]
```

2. 请参考 ch24_6.py 绘制 n_trials 是 50 次时，p=0.2, 0.5, 0.8 时二项分布之概率累积函数图。（**24-2 节**）



3. 请重新设计 ch24_10.py，将标准偏差改为 1.5，同时计算落在 −1 和 1 之间的概率值。（**24-2 节**）

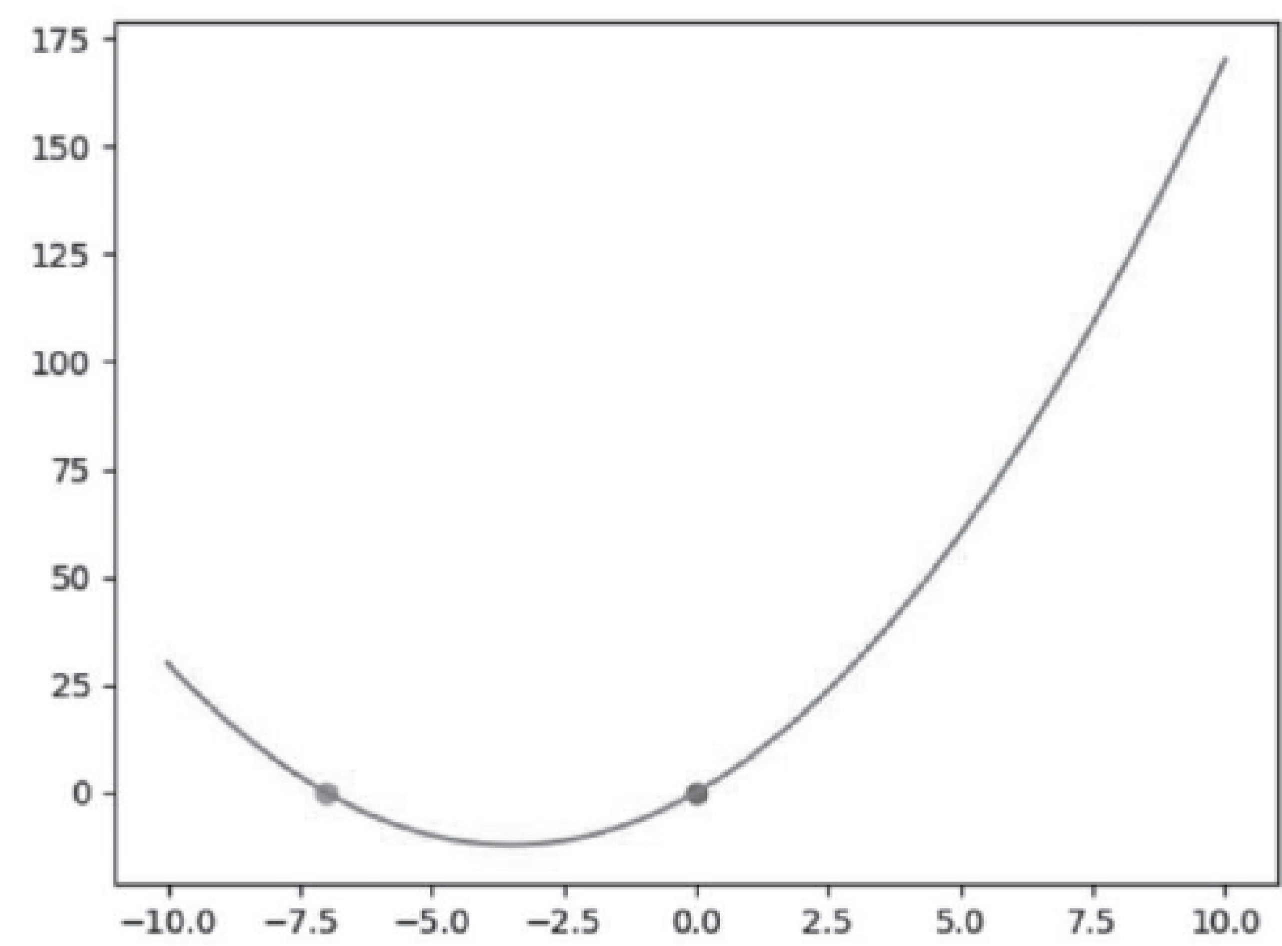
```
====================== RESTART: D:\Python\ex\ex24_3.py ======================
落在-1与1之间的概率是 49.50%
```



4. 计算下列一元二次方程式的根，同时绘制此函数图形。（**24-3 节**）

$x^2 + 7x = 0$

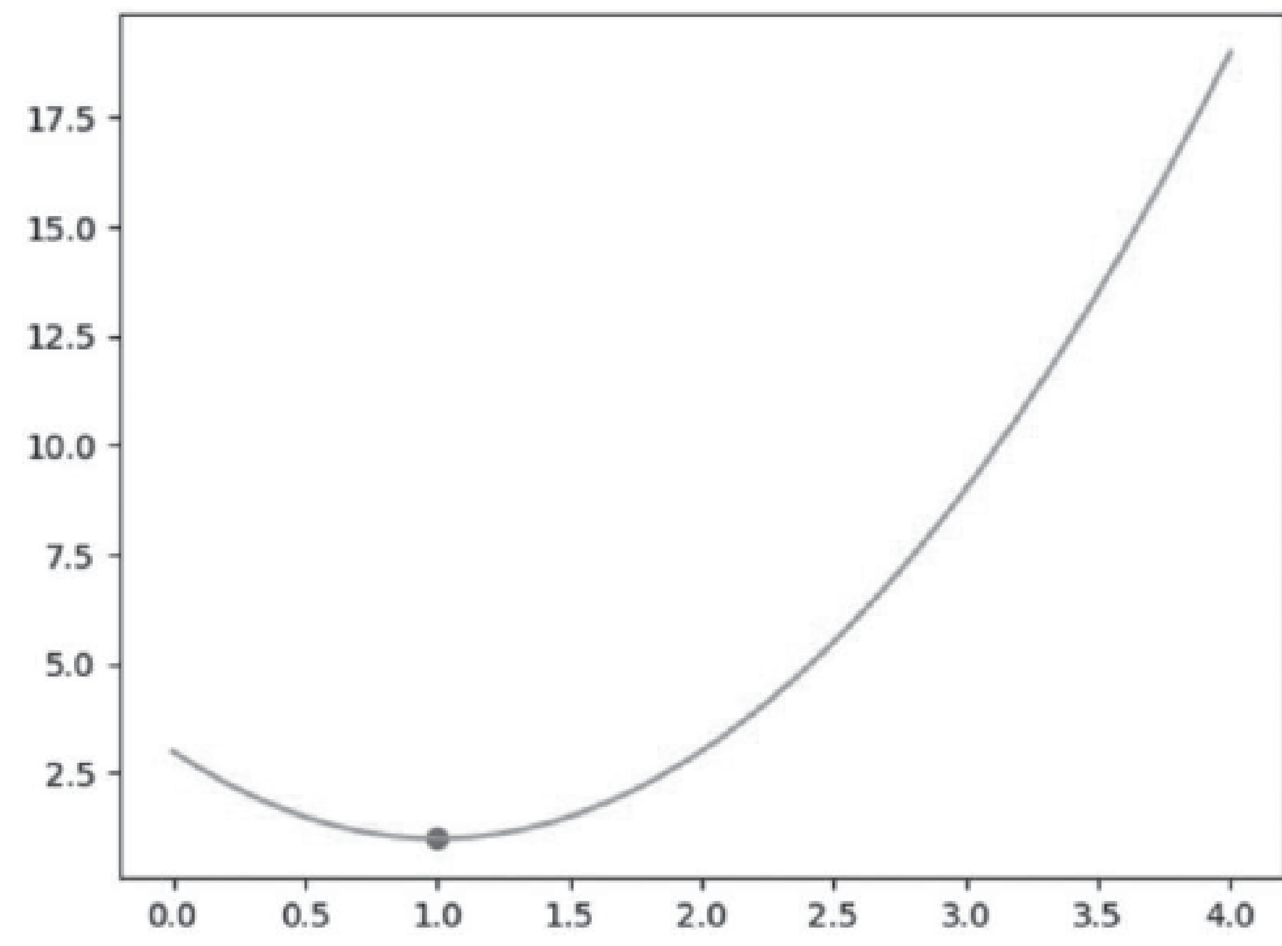
```
====================== RESTART: D:/Python/ex/ex24_4.py ======================
[0.]
[-7.]
```



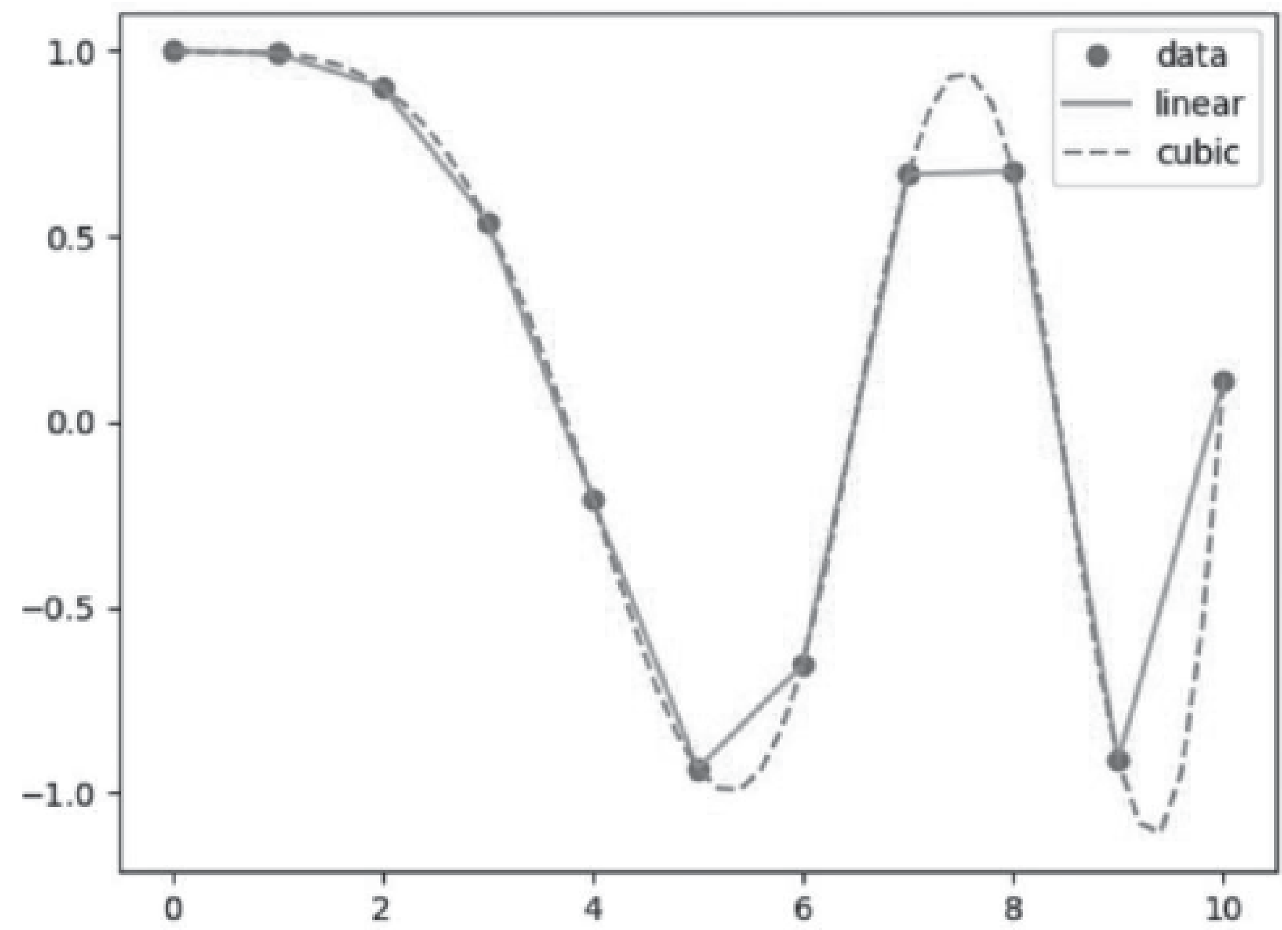
5. 找出下列函数的最小值与其坐标，同时绘制此函数图形。（**24-3 节**）

$f(x) = 2(x-2)^2 + 4x - 5$

```
===================== RESTART: D:\Python\ex\ex24_5.py =====================
当x是 1.00 时，有函数最小值
坐标是（1.00，1.00）
```



6. 请修改程序实例 ch24_19.py，将原始数据函数改为 cos（$x^2/9$），同时最初在 1 ～ 10 取 11 个点，在执行 linear 和 cubic 插入时，在 0 ～ 10 取 51 个点。（**24-4 节**）



25

第 25 章

Pandas 模块

本章摘要

Pandas 是一个建构在 Numpy 之上，专为 Python 编写的外部模块，主要是整合了 Numpy、Scipy 和 Matplotlab 的功能，可以很方便地执行数据处理与分析。它的名称主要是来自 **pan**el、**da**taframe 与 **s**eries，而这 3 个单词也是 Pandas 的 3 个数据结构 Panel、DataFrame 和 Series。

有时候 Panda 也被称为“熊猫”，使用此模块前请使用下列方式安装：

```
pip install panda
```

安装完成后可以使用下列方式导入模块，了解目前的 Pandas 版本。

```
>>> import pandas as pd
>>> pd.__version__
'0.24.1'
```

本章将介绍 Pandas 最基础与最常用的部分，读者若想了解更多可以参考下列网址：

https://pandas.pydata.org

25-1 Series

Series 是一种一维的数组数据结构，在这个数组内可以存放整数、浮点数、字符串、Python 对象（例如字符串 list、字典 dist 等）以及 Numpy 的 ndarray、纯量等。虽然是一维数组数据，可是看起来却好像是二维数组数据，因为一个是索引（index）或称标签（label），另一个是实际的数据。

Series 结构与 Python 的 list 类似，不过程序设计师可以为 Series 的每个元素自行命名索引。可以使用 pd.Series( ) 建立 Series 对象，语法如下：

```
pandas.Series(data=None, index=None, dtype=None, name=None, options, …)
```

25-1-1 使用列表 list 建立 Series 对象

最简单的建立 Series 对象的方式是在 data 参数使用列表。

实例 1：在 data 参数使用列表建立 Series 对象 s1，然后列出结果。

```
>>> import pandas as pd
>>> s1 = pd.Series([11,22,33,44,55])
>>> s1
0    11
1    22
2    33
3    44
4    55
dtype: int64
```

我们只有建立 Series 对象 s1 内容，可是打印时看到左边字段有系统自建的索引，Pandas 的索引也是从 0 开始计数，这也是为什么我们说 Series 是一个一维数组，可是看起来像是二维数组的原因。有了这个索引，可以使用索引存取对象内容。

实例 2：延续先前实例，列出 Series 特定索引内容与修改内容。

```
>>> s1[1]
22
>>> s1[1] = 20
>>> s1
0    11
1    20
2    33
3    44
4    55
dtype: int64
```

25-1-2　使用 Python 字典 dict 建立 Series 对象

如果我们使用 Python 的字典建立 Series 对象时，字典的键（key）就会被视为 Series 对象的索引，字典键的值（value）就会被视为 Series 对象的值。

实例：使用 Python 字典 dict 建立 Series 对象，同时列出结果。

```
>>> import pandas as pd
>>> mydict = {'北京':'Beijing', '东京':'Tokey'}
>>> s2 = pd.Series(mydict)
>>> s2
北京    Beijing
东京      Tokey
dtype: object
```

25-1-3　使用 Numpy 的 ndarray 建立 Series 对象

实例：使用 Numpy 的 ndarray 建立 Series 对象，同时列出结果。

```
>>> import pandas as pd
>>> import numpy as np
>>> s3 = pd.Series(np.arange(0, 7, 2))
>>> s3
0    0
1    2
2    4
3    6
dtype: int32
```

25-1-4　建立含索引的 Series 对象

目前为止我们了解在建立 Series 对象时，默认情况索引是从 0 开始计数，若是我们使用字典建立 Series 对象，字典的健（key）就是索引。其实在建立 Series 对象时，也可以使用 index 参数自行建立索引。

实例 1：建立索引不是从 0 开始计数。

```
>>> myindex = [3, 5, 7]
>>> price = [100, 200, 300]
>>> s4 = pd.Series(price, index=myindex)
>>> s4
3    100
5    200
7    300
dtype: int64
```

实例 2：建立含自定义索引的 Series 对象，同时列出结果。

```
>>> fruits = ['Orange', 'Apple', 'Grape']
>>> price = [30, 50, 40]
>>> s5 = pd.Series(price, index=fruits)
>>> s5
Orange    30
Apple     50
Grape     40
dtype: int64
```

上述有时候也可以用下列方式建立一样的 Series 对象。

```
s5 = pd.Series([30, 50, 40], index=['Orange', 'Apple', 'Grape'])
```

由上述内容读者应该体会到，Series 对象有一个很大的特色，那就是可以使用任意方式的索引。

25-1-5　使用纯量建立 Series 对象

实例：使用纯量建立 Series 对象，同时列出结果。

```
>>> s6 = pd.Series(9, index=[1,2,3])
>>> s6
1    9
2    9
3    9
dtype: int64
```

虽然只有一个纯量搭配 3 个索引，Pandas 会主动将所有索引值用此纯量补上。

25-1-6　列出 Series 对象索引与值

从前面实例可以知道，我们可以直接用 print（对象名称），打印 Series 对象，其实也可以使用下列方式得到 Series 对象索引和值：

```
obj.values              # 假设对象名称是 obj，Series 对象值
obj.index               # 假设对象名称是 obj，Series 对象索引
```

实例：打印 Series 对象索引和值。

```
>>> s5 = pd.Series([30, 50, 40], index=['Orange', 'Apple', 'Grape'])
>>> print(s5.values)
[30 50 40]
>>> print(s5.index)
Index(['Orange', 'Apple', 'Grape'], dtype='object')
```

25-1-7　Series 的运算

Series 的运算方法许多与 Numpy 的 ndarray 或是 Python 的列表相同，但是有一些更好用的功能，本小节会做解说。

实例 1：可以将切片概念应用在 Series 对象。

```
>>> s = pd.Series([0, 1, 2, 3, 4, 5])
>>> s[2:4]
2    2
3    3
dtype: int64
>>> s[:3]
0    0
1    1
2    2
dtype: int64
>>> s[2:]
2    2
3    3
4    4
5    5
dtype: int64
>>> s[-1:]
5    5
dtype: int64
```

四则运算与求余数的概念也可以应用在 Series 对象。

实例 2：Series 物件相加。

```
>>> x = pd.Series([1, 2])
>>> y = pd.Series([3, 4])
>>> x + y
0    4
1    6
dtype: int64
```

实例 3：Series 物件相乘。

```
>>> x = pd.Series([1, 2])
>>> y = pd.Series([3, 4])
>>> x * y
0    3
1    8
dtype: int64
```

逻辑运算的概念也可以应用在 Series 对象。

实例 4：逻辑运算应用在 Series 对象。

```
>>> x = pd.Series([1, 5, 9])
>>> y = pd.Series([2, 4, 8])
>>> x > y
0    False
1     True
2     True
dtype: bool
```

有两个 Series 对象拥有相同的索引，这时也可以将这两个对象相加。

实例 5：Series 对象拥有相同索引，执行相加的应用。

```
>>> fruits = ['Orange', 'Apple', 'Grape']
>>> x1 = pd.Series([20, 30, 40], index=fruits)
>>> x2 = pd.Series([25, 38, 55], index=fruits)
>>> y = x1 + x2
>>> y
Orange    45
Apple     68
Grape     95
dtype: int64
```

在执行相加时，如果两个索引不相同，也可以执行相加，这时不同索引的索引内容值会填上 NaN（Not a Number），可以解释为非数字或无定义数字。

实例 6：Series 对象拥有不同索引，执行相加的应用。

```
>>> fruits1 = ['Orange', 'Apple', 'Grape']
>>> fruits2 = ['Orange', 'Banana', 'Grape']
>>> x1 = pd.Series([20, 30, 40], index=fruits1)
>>> x2 = pd.Series([25, 38, 55], index=fruits2)
>>> y = x1 + x2
>>> y
Apple      NaN
Banana     NaN
Grape     95.0
Orange    45.0
dtype: float64
```

当索引是非数值而是字符串时，可以使用下列方式取得元素内容。

实例 7：Series 的索引是字符串，取得元素内容的应用。

```
>>> fruits = ['Orange', 'Apple', 'Grape']
>>> x = pd.Series([20, 30, 40], index=fruits)
>>> print(x['Apple'])
30
>>> print(x[['Apple', 'Orange']])
Apple     30
Orange    20
dtype: int64
>>> print(x[['Orange', 'Apple', 'Grape']])
Orange    20
Apple     30
Grape     40
dtype: int64
```

我们也可以将纯量与 Series 对象做运算，甚至也可以将函数应用在 Series 对象。

实例 8：将纯量与和函数应用在 Series 对象上。

```
>>> fruits = ['Orange', 'Apple', 'Grape']
>>> x = pd.Series([20, 30, 40], index=fruits)
>>> print((x + 10) * 2)
Orange     60
Apple      80
Grape     100
dtype: int64
>>> print(np.sin(x))
Orange    0.912945
Apple    -0.988032
Grape     0.745113
dtype: float64
```

25-2 DataFrame

DataFrame 是一种二维的数组数据结构，逻辑上而言可以视为是类似 Excel 的工作表，在这个二维数组内可以存放整数、浮点数、字符串、Python 对象（例如字符串 list、字典 dist 等）以及 Numpy 的 ndarray、纯量等。

可以使用 DataFrame( ) 建立 DataFrame 对象，语法如下：

```
pandas.DataFrame(data=None,index=None,dtype=None,name=None)
```

25-2-1 建立 DataFrame 使用 Series

我们可以使用组合 Series 对象成为二维数组的 DataFrame。组合的方式是使用 pandas.concat ([Series1, Series2, …], axis=1)。

程序实例 ch25_1.py：建立 Beijing、HongKong、Singapore 2020—2022 年 3 月的平均温度，成为 3 个 Series 对象。笔者设置 concat( ) 方法不设置 axis，结果不是我们预期。

```
1  # ch25_1.py
2  import pandas as pd
3  years = range(2020, 2023)
4  beijing = pd.Series([20, 21, 19], index = years)
5  hongkong = pd.Series([25, 26, 27], index = years)
6  singapore = pd.Series([30, 29, 31], index = years)
7  citydf = pd.concat([beijing, hongkong, singapore])  # 预设axis=0
8  print(type(citydf))
9  print(citydf)
```

执行结果

```
==================== RESTART: D:\Python\ch25\ch25_1.py ====================
<class 'pandas.core.series.Series'>
2020    20
2021    21
2022    19
2020    25
2021    26
2022    27
2020    30
2021    29
2022    31
dtype: int64
```

很明显上述不是我们的预期，经过 concat( ) 方法组合后，citydf 数据形态仍是 Series，问题出现在使用 concat( ) 组合 Series 对象时 axis 的默认是 0，如果将第 7 行改为增加 axis=1 参数即可。

程序实例 ch25_2.py：重新设计 ch25_1.py 建立 DataFrame 对象。

```
1  # ch25_2.py
2  import pandas as pd
3  years = range(2020, 2023)
4  beijing = pd.Series([20, 21, 19], index = years)
5  hongkong = pd.Series([25, 26, 27], index = years)
6  singapore = pd.Series([30, 29, 31], index = years)
7  citydf = pd.concat([beijing, hongkong, singapore],axis=1)  # axis=1
8  print(type(citydf))
9  print(citydf)
```

执行结果

```
==================== RESTART: D:/Python/ch25/ch25_2.py ====================
<class 'pandas.core.frame.DataFrame'>
       0   1   2
2020  20  25  30
2021  21  26  29
2022  19  27  31
```

从上述执行结果我们已经得到所要的 DataFrame 对象了。

25-2-2　字段 columns 属性

上述 ch25_2.py 的执行结果不完美是因为字段 columns 没有名称，在 pandas 中可以使用 columns 属性设置域名。

程序实例 ch25_3.py：扩充 ch25_2.py，使用 columns 属性设置域名。

```
1  # ch25_3.py
2  import pandas as pd
3  years = range(2020, 2023)
4  beijing = pd.Series([20, 21, 19], index = years)
5  hongkong = pd.Series([25, 26, 27], index = years)
6  singapore = pd.Series([30, 29, 31], index = years)
7  citydf = pd.concat([beijing, hongkong, singapore],axis=1)  # axis=1
8  cities = ["Beijing", "HongKong", "Singapore"]
9  citydf.columns = cities
10 print(citydf)
```

执行结果

```
==================== RESTART: D:/Python/ch25/ch25_3.py ====================
      Beijing  HongKong  Singapore
2020       20        25         30
2021       21        26         29
2022       19        27         31
```

25-2-3　Series 对象的 name 属性

Series 对象有 name 属性，我们可以在建立对象时，在 Series( ) 内建立此属性，也可以等对象建立好了后再设置此属性，如果有 name 属性，在打印 Series 对象时就可以看到此属性。

实例：建立 Series 对象时，同时建立 name。

```
>>> beijing = pd.Series([20, 21, 19], name='Beijing')
>>> beijing
0    20
1    21
2    19
Name: Beijing, dtype: int64
```

程序实例 ch25_4.py：更改 ch25_3.py 的设计方式，使用 name 属性设置 DataFrame 的 columns 域名。

```
1  # ch25_4.py
2  import pandas as pd
3  years = range(2020, 2023)
4  beijing = pd.Series([20, 21, 19], index = years)
5  hongkong = pd.Series([25, 26, 27], index = years)
6  singapore = pd.Series([30, 29, 31], index = years)
7  beijing.name = "Beijing"
8  hongkong.name = "HongKong"
9  singapore.name = "Singapore"
10 citydf = pd.concat([beijing, hongkong, singapore],axis=1)
11 print(citydf)
```

执行结果 与 ch25_3.py 相同。

25-2-4　使用元素是字典的列表建立 DataFrame

有一个列表它的元素是字典时，可以使用此列表建立 DataFrame。

程序实例 ch25_5.py：使用元素是字典的列表建立 DataFrame 对象。

```
1  # ch25_5.py
2  import pandas as pd
3  data = [{'apple':50,'Orange':30,'Grape':80},{'apple':50,'Grape':80}]
4  fruits = pd.DataFrame(data)
5  print(fruits)
```

执行结果

```
==================== RESTART: D:/Python/ch25/ch25_5.py ====================
   Grape  Orange  apple
0     80    30.0     50
1     80     NaN     50
```

上述如果碰上字典健（key）没有对应，该位置将填入 NaN。

25-2-5　使用字典建立 DataFrame

一个字典健（key）的值（value）是列表时，也可以很方便地用于建立 DataFrame。

程序实例 ch25_6.py：使用字典建立 DataFrame 对象。

```
1  # ch25_6.py
2  import pandas as pd
3  cities = {'country':['China', 'Japan', 'Singapore'],
4            'town':['Beijing','Tokyo','Singapore'],
5            'population':[2000, 1600, 600]}
6  citydf = pd.DataFrame(cities)
7  print(citydf)
```

执行结果

```
==================== RESTART: D:\Python\ch25\ch25_6.py ====================
     country       town  population
0      China    Beijing        2000
1      Japan      Tokyo        1600
2  Singapore  Singapore         600
```

25-2-6　index 属性

对于 DataFrame 对象而言，我们可以使用 index 属性设置对象的 row 卷标，例如，若是以 ch25_6.py 的执行结果而言，0,1,2 索引就是 row 卷标。

程序实例 ch25_7.py：重新设计 ch25_6.py，将 row 标签改为 first, second, third。

```
1  # ch25_7.py
2  import pandas as pd
3  cities = {'country':['China', 'Japan', 'Singapore'],
4            'town':['Beijing','Tokyo','Singapore'],
5            'population':[2000, 1600, 600]}
6  rowindex = ['first', 'second', 'third']
7  citydf = pd.DataFrame(cities, index=rowindex)
8  print(citydf)
```

执行结果

```
==================== RESTART: D:\Python\ch25\ch25_7.py ====================
          country       town  population
first       China    Beijing        2000
second      Japan      Tokyo        1600
third   Singapore  Singapore         600
```

25-2-7　将 columns 字段当作 DataFrame 对象的 index

另外，以字典方式建立 DataFrame，如果字典内某个元素被当作 index 时，这个元素就不会在 DataFrame 的字段 columns 上出现。

程序实例 ch25_8.py：重新设计 ch25_7.py，这个程序会将 country 当作 index。

```
1  # ch25_8.py
2  import pandas as pd
3  cities = {'country':['China', 'Japan', 'Singapore'],
4            'town':['Beijing','Tokyo','Singapore'],
5            'population':[2000, 1600, 600]}
6  citydf = pd.DataFrame(cities, columns=["town","population"],
7                        index=cities["country"])
8  print(citydf)
```

执行结果

```
==================== RESTART: D:/Python/ch25/ch25_8.py ====================
                town  population
China        Beijing        2000
Japan          Tokyo        1600
Singapore  Singapore         600
```

25-3 基本 Pandas 数据分析与处理

Series 和 DataFrame 对象建立完成后，下一步就是执行数据分析与处理，Pandas 提供了许多函数或方法，用户可以执行许多数据分析与处理，本节将讲解基本概念，读者若想更进一步学习可以参考 Pandas 专业书籍，或是参考 Pandas 官方网站。

25-3-1 索引参照属性

本小节将说明下列属性的用法：

at：使用 index 和 columns 内容取得或设置单一元素内容或数组内容。

iat：使用 index 和 columns 编号取得或设置单一元素内容。

loc：使用 index 或 columns 内容取得或设置整个 row 或 columns 数据或数组内容。

iloc：使用 index 或 columns 编号取得或设置整个 row 或 columns 数据。

程序实例 ch25_9.py：在说明上述属性用法前，笔者先建立一个 DataFrame 对象，然后用此对象做解说。

```
1  # ch25_9.py
2  import pandas as pd
3  cities = {'Country':['China','China','Thailand','Japan','Singapore'],
4            'Town':['Beijing','Shanghai','Bangkok', 'Tokyo','Singapore'],
5            'Population':[2000, 2300, 900, 1600, 600]}
6  df = pd.DataFrame(cities, columns=["Town","Population"],
7                    index=cities["Country"])
8  print(df)
```

执行结果 下列是 Python Shell 窗口的执行结果，下列实例请在此窗口执行。

```
==================== RESTART: D:/Python/ch25/ch25_9.py ====================
                Town  Population
China        Beijing        2000
China       Shanghai        2300
Thailand     Bangkok         900
Japan          Tokyo        1600
Singapore  Singapore         600
```

实例 1：使用 at 属性 row 是 'Japan'，column 是 'Town'，并列出结果。

```
>>> df.at['Japan','Town']
'Tokyo'
```

如果观察可以看到有两个索引是 'China'，如果 row 是 'China' 时，这时可以获得数组数据，可以参考下列实例。

实例 2：使用 at 属性取得 row 是 'China'，column 是 'Town'，并列出结果。

```
>>> df.at['China', 'Town']
array(['Beijing', 'Shanghai'], dtype=object)
```

实例 3：使用 iat 属性取得 row 是 2，column 是 0，并列出结果。

```
>>> df.iat[2,0]
'Bangkok'
```

实例 4：使用 loc 属性取得 row 是 'Singapore'，并列出结果。

```
>>> df.loc['Singapore']
Town          Singapore
Population          600
Name: Singapore, dtype: object
```

实例 5：使用 loc 属性取得 row 是 'Japan' 和 'Thailand'，并列出结果。

```
>>> df.loc[['Japan', 'Thailand']]
             Town  Population
Japan       Tokyo        1600
Thailand  Bangkok         900
```

实例 6：使用 loc 属性取得 row 是 'China':'Thailand'，column 是 'Town':'Population'，并列出结果。

```
>>> df.loc['China':'Thailand','Town':'Population']
              Town  Population
China      Beijing        2000
China     Shanghai        2300
Thailand   Bangkok         900
```

实例 7：使用 iloc 属性取得 row 是 0 的数据，并列出结果。

```
>>> df.iloc[0]
Town          Beijing
Population       2000
Name: China, dtype: object
```

25-3-2　直接索引

除了上一节的方法可以取得 DataFrame 的对象内容，也可以使用直接索引方式取得内容，这一小节仍将继续使用 ch25_9.py 所建的 DataFrame 物件 df。

实例 1：直接索引取得 'Town' 的数据并打印。

```
>>> df['Town']
China          Beijing
China         Shanghai
Thailand       Bangkok
Japan            Tokyo
Singapore    Singapore
Name: Town, dtype: object
```

实例 2：取得 column 是 'Town'，row 是 'Japan' 的数据并打印。

```
>>> df['Town']['Japan']
'Tokyo'
```

实例 3：取的 column 是 'Town' 和 'Population' 的数据并打印。

```
>>> df[['Town','Population']]
                Town  Population
China        Beijing        2000
China       Shanghai        2300
Thailand     Bangkok         900
Japan          Tokyo        1600
Singapore  Singapore         600
```

实例 4：取得 row 编号 3 之前的数据并打印。

```
>>> df[:3]
              Town  Population
China      Beijing        2000
China     Shanghai        2300
Thailand   Bangkok         900
```

实例 5：取得 Population 大于 1000 的数据并打印。

```
>>> df[df['Population'] > 1000]
           Town  Population
China   Beijing        2000
China  Shanghai        2300
Japan     Tokyo        1600
```

25-3-3 四则运算方法

下列是适用 Pandas 的四则运算方法：

add( )：加法运算。

sub( )：减法运算。

mul( )：乘法运算。

div( )：除法运算。

实例 1：加法与减法运算。

```
>>> s1 = pd.Series([1,2,3])
>>> s2 = pd.Series([4,5,6])
>>> x = s1.add(s2)
>>> print(x)
0    5
1    7
2    9
dtype: int64
>>> y = s1.sub(s2)
>>> print(y)
0   -3
1   -3
2   -3
dtype: int64
```

实例 2：乘法与除法运算。

```
>>> data1 = [{'a':10,'b':20}, {'a':30, 'b':40}]
>>> df1 = pd.DataFrame(data1)
>>> data2 = [{'a':1,'b':2}, {'a':3, 'b':4}]
>>> df2 = pd.DataFrame(data2)
>>> x = df1.mul(df2)
>>> print(x)
    a    b
0  10   40
1  90  160
>>> y = df1.div(df2)
>>> print(y)
      a     b
0  10.0  10.0
1  10.0  10.0
```

25-3-4 逻辑运算方法

下列是适用 Pandas 的逻辑运算方法：

gt( )、lt( )：大于、小于运算。

ge( )、le( )：大于或等于、小于或等于运算。

eq( )、ne( )：等于、不等于运算。

实例：逻辑运算 gt( ) 和 eq( ) 的应用。

```
>>> s1 = pd.Series([1,5,9])
>>> s2 = pd.Series([2,4,8])
>>> x = s1.gt(s2)
>>> print(x)
0    False
1     True
2     True
dtype: bool
>>> y = s1.eq(s2)
>>> print(y)
0    False
1    False
2    False
dtype: bool
```

25-3-5　Numpy 的函数应用在 Pandas

实例：将 Numpy 的函数 square( ) 应用在 Series。

```
>>> import numpy as np
>>> import pandas as pd
>>> s = pd.Series([1,2,3])
>>> x = np.square(s)
>>> print(x)
0    1
1    4
2    9
dtype: int64
```

程序实例 ch25_10.py：将 Numpy 的随机值函数 randint( ) 应用在建立 DataFrame 对象的元素内容，假设有一门课程，第一次 first、第二次 second 和最后成绩 final 皆是使用随机数给予，分数是 60 ～ 99。

```
1  # ch25_10.py
2  import pandas as pd
3  import numpy as np
4  name = ['Frank', 'Peter', 'John']
5  score = ['first', 'second', 'final']
6  df = pd.DataFrame(np.random.randint(60,100,size=(3,3)),
7                    columns=name,
8                    index=score)
9  print(df)
```

执行结果

```
==================== RESTART: D:/Python/ch25/ch25_10.py ====================
        Frank  Peter  John
first      86     60    76
second     76     76    88
final      96     70    99
```

25-3-6　NaN 相关的运算

在大数据的数据收集中，常常因为执行者疏忽，漏了收集某一时间的数据，这些可用 NaN 代替。在先前四则运算中，我们没有对 NaN 的值做运算实例，其实凡与 NaN 做运算，所获得的结果也是 NaN。

实例：与 NaN 相关的运算

```
>>> s1 = pd.Series([1, np.nan, 5])
>>> s2 = pd.Series([np.nan, 6, 8])
>>> x = s1.add(s2)
>>> print(x)
0     NaN
1     NaN
2    13.0
dtype: float64
```

25-3-7 NaN 的处理

下列是适合处理 NaN 的方法：

dropna( )：将 NaN 删除，然后返回新的 Series 或 DataFrame 对象。

fillna(value)：将 NaN 由特定 value 值取代，然后返回新的 Series 或 DataFrame 对象。

isna( )：判断是否为 NaN，如果是返回 True，如果否返回 False。

notna( )：判断是否为 NaN，如果是返回 False，如果否返回 True。

实例 1：isna( ) 和 notna( ) 的应用。

```
>>> df = pd.DataFrame([[1,2,3],[4,np.nan,6],[7,8,np.nan]])
>>> df
   0    1    2
0  1  2.0  3.0
1  4  NaN  6.0
2  7  8.0  NaN
>>> x = df.isna()
>>> print(x)
       0      1      2
0  False  False  False
1  False   True  False
2  False  False   True
>>>
>>> y = df.notna()
>>> print(y)
      0      1      2
0  True   True   True
1  True  False   True
2  True   True  False
```

实例 2：沿用先前实例在 NaN 位置填上 0。

```
>>> z = df.fillna(0)
>>> print(z)
   0    1    2
0  1  2.0  3.0
1  4  0.0  6.0
2  7  8.0  0.0
```

实例 3：dropna( ) 如果不含参数，会删除含 NaN 的 row。

```
>>> a = df.dropna()
>>> print(a)
   0    1    2
0  1  2.0  3.0
```

实例 4：删除含 NaN 的 columns。

```
>>> b = df.dropna(axis='columns')
>>> print(b)
   0
0  1
1  4
2  7
```

25-3-8　几个简单的统计函数

cummax(axis=None)：返回指定轴累积的最大值。

cummin(axis=None)：返回指定轴累积的最小值。

cumsum(axis=None)：返回指定轴累积的总和。

max(axis=None)：返回指定轴的最大值。

min(axis=None)：返回指定轴的最小值。

sum(axis=None)：返回指定轴的总和。

mean(axis=None)：返回指定轴的平均数。

median(axis=None)：返回指定轴的中位数。

std(axis=None)：返回指定轴的标准偏差。

实例 1：请再执行一次 ch25_9.py，方便取得 DataFrame 对象 df 的数据，然后使用此数据，列出这些城市的人口总计 sum( ) 和累积人口总计 cumsum( )。

```
===================== RESTART: D:\Python\ch25\ch25_9.py =====================
                Town  Population
China        Beijing        2000
China       Shanghai        2300
Thailand     Bangkok         900
Japan          Tokyo        1600
Singapore  Singapore         600
>>> x = df['Population'].sum()
>>> print(x)
7400
>>> y = df['Population'].cumsum()
>>> print(y)
China        2000
China        4300
Thailand     5200
Japan        6800
Singapore    7400
Name: Population, dtype: int64
```

实例 2：延续前一个实例，在 df 对象内插入人口累积总数 Sum_Population 字段。

```
>>> df['Cum_Population'] = y
>>> print(df)
                Town  Population  Cum_Population
China        Beijing        2000            2000
China       Shanghai        2300            4300
Thailand     Bangkok         900            5200
Japan          Tokyo        1600            6800
Singapore  Singapore         600            7400
```

实例 3：列出最多与最小人口数。

```
>>> df['Population'].max()
2300
>>> df['Population'].min()
600
```

程序实例 ch25_11.py：有几位学生大学学测分数如下：

	语文	英文	数学	自然	社会
1	14	13	15	15	12
2	12	14	9	10	11
3	13	11	12	13	14
4	10	10	8	10	9
5	13	15	15	15	14

请建立此 DataFrame 对象，同时打印。

```
1  # ch25_11.py
2  import pandas as pd
3
4  course = ['Chinese', 'English', 'Math', 'Natural', 'Society']
5  chinese = [14, 12, 13, 10, 13]
6  eng = [13, 14, 11, 10, 15]
7  math = [15, 9, 12, 8, 15]
8  nature = [15, 10, 13, 10, 15]
9  social = [12, 11, 14, 9, 14]
10
11 df = pd.DataFrame([chinese, eng, math, nature, social],
12                   columns = course,
13                   index = range(1,6))
14 print(df)
```

执行结果

```
==================== RESTART: D:\Python\ch25\ch25_11.py ====================
   Chinese  English  Math  Natural  Society
1       14       12    13       10       13
2       13       14    11       10       15
3       15        9    12        8       15
4       15       10    13       10       15
5       12       11    14        9       14
```

实例 4：列出每位学生的总分。

```
>>> total = [df.iloc[i].sum() for i in range(0, 5)]
>>> print(total)
[62, 63, 59, 63, 60]
```

实例 5：增加总分字段，然后列出 DataFrame。

```
>>> df['Total'] = total
>>> print(df)
   Chinese  English  Math  Natural  Society  Total
1       14       12    13       10       13     62
2       13       14    11       10       15     63
3       15        9    12        8       15     59
4       15       10    13       10       15     63
5       12       11    14        9       14     60
```

实例 6：列出各科平均分数，同时也列出平均分数的总分。

```
>>> ave = df.mean()
>>> print(ave)
Chinese     13.8
English     11.2
Math        12.6
Natural      9.4
Society     14.4
Total       61.4
dtype: float64
```

25-3-9 增加 index

可以使用 loc 属性为 DataFrame 增加平均分数。

实例：在 df 下方增加 Average 平均分数。

```
>>> df.loc['Average'] = ave
>>> print(df)
         Chinese  English  Math  Natural  Society  Total
1           14.0     12.0  13.0     10.0     13.0   62.0
2           13.0     14.0  11.0     10.0     15.0   63.0
3           15.0      9.0  12.0      8.0     15.0   59.0
4           15.0     10.0  13.0     10.0     15.0   63.0
5           12.0     11.0  14.0      9.0     14.0   60.0
Average     13.8     11.2  12.6      9.4     14.4   61.4
```

25-3-10　删除 index

若是想删除 index 是 Average，可以使用 drop( )，可以参考下列实例。

实例：删除 Average。

```
>>> df = df.drop(index=['Average'])
>>> print(df)
  Chinese  English  Math  Natural  Society  Total
1    14.0     12.0  13.0     10.0     13.0   62.0
2    13.0     14.0  11.0     10.0     15.0   63.0
3    15.0      9.0  12.0      8.0     15.0   59.0
4    15.0     10.0  13.0     10.0     15.0   63.0
5    12.0     11.0  14.0      9.0     14.0   60.0
```

25-3-11　排序

排序可以使用 sort_values( )，可以参考下列实例。

实例 1：将 DataFrame 对象 Total 字段从大排到小。

```
>>> df = df.sort_values(by='Total', ascending=False)
>>> print(df)
  Chinese  English  Math  Natural  Society  Total
2    13.0     14.0  11.0     10.0     15.0   63.0
4    15.0     10.0  13.0     10.0     15.0   63.0
1    14.0     12.0  13.0     10.0     13.0   62.0
5    12.0     11.0  14.0      9.0     14.0   60.0
3    15.0      9.0  12.0      8.0     15.0   59.0
```

上述预设是从小排到大，所以 sort_values( ) 增加参数 ascending=False，改为从大排到小。

实例 2：增加名次字段，然后填入名次（Ranking）。

```
>>> rank = range(1,6)
>>> df['Ranking'] = rank
>>> print(df)
  Chinese  English  Math  Natural  Society  Total  Ranking
2    13.0     14.0  11.0     10.0     15.0   63.0        1
4    15.0     10.0  13.0     10.0     15.0   63.0        2
1    14.0     12.0  13.0     10.0     13.0   62.0        3
5    12.0     11.0  14.0      9.0     14.0   60.0        4
3    15.0      9.0  12.0      8.0     15.0   59.0        5
```

上述有一个地方不完美，第 2 行与第 1 行的总分一样是 63 分，但是名次是第 2 名，我们可以使用下列方式解决。

实例 3：设置同分数应该有相同名次。

```
>>> for i in range(1,5):
        if df.iat[i,5] == df.iat[i-1,5]:
            df.iat[i,6] = df.iat[i-1,6]

>>> print(df)
  Chinese  English  Math  Natural  Society  Total  Ranking
2    13.0     14.0  11.0     10.0     15.0   63.0        1
4    15.0     10.0  13.0     10.0     15.0   63.0        1
1    14.0     12.0  13.0     10.0     13.0   62.0        3
5    12.0     11.0  14.0      9.0     14.0   60.0        4
3    15.0      9.0  12.0      8.0     15.0   59.0        5
```

实例 4：依 index 重新排序，这时可以使用 sort_index( )。

```
>>> df = df.sort_index()
>>> print(df)
  Chinese  English  Math  Natural  Society  Total  Ranking
1    14.0     12.0  13.0     10.0     13.0   62.0        3
2    13.0     14.0  11.0     10.0     15.0   63.0        1
3    15.0      9.0  12.0      8.0     15.0   59.0        5
4    15.0     10.0  13.0     10.0     15.0   63.0        1
5    12.0     11.0  14.0      9.0     14.0   60.0        4
```

25-4 文件的输入与输出

Pandas 可以读取的文件有许多，如 TXT、CSV、JSON、Excel 等，也可以将文件以上述资料格式写入文件。本节将说明读写 CSV 格式的文件。

CSV 是一个缩写，它的英文全名是 Comma-Separated Values，由字面意义可以理解为“逗号分隔值”，当然逗号是主要数据字段间的分隔值，不过目前也有非逗号的分隔值。这是一个纯文本格式的文件，没有图片，也不用考虑字形、大小、颜色等。

简单地说，CSV 数据是指同一行的资料彼此用逗号（或其他符号）隔开，同时每一行数据是一笔（record）数据。几乎所有电子表格与数据库文件均支持这个格式，所以也可以用 Excel 打开此文件。

25-4-1 写入 CSV 格式文件

Pandas 可以使用 to_csv( ) 将 DataFrame 对象写入 CSV 文件，它的语法如下：

to_csv (path=None, sep=',', header=True, index=True, encoding=None, …)

path：文件路径（名称）。

sep：分隔字符，默认是‘,’。

header：是否保留 columns，预设是 True。

index：是否保留 index，预设是 True。

encoding：文件编码方式。

程序实例 ch25_12.py：将 ch25_11.py 所建立的 DataFrame 对象，用有保留 header 和 index 方式存储至 out15_12a.csv，然后也用没有保留的方式存入 out15_12b.csv。

```
1  # ch25_12.py
2  import pandas as pd
3  import numpy as np
4
5  course = ['Chinese', 'English', 'Math', 'Natural', 'Society']
6  chinese = [14, 12, 13, 10, 13]
7  eng = [13, 14, 11, 10, 15]
8  math = [15, 9, 12, 8, 15]
9  nature = [15, 10, 13, 10, 15]
10 social = [12, 11, 14, 9, 14]
11
12 df = pd.DataFrame([chinese, eng, math, nature, social],
13                   columns = course,
14                   index = range(1,6))
15 df.to_csv("out25_12a.csv")
16 df.to_csv("out25_12b.csv", header=False, index=False)
```

执行结果

下列是 out25_12a.csv 与 out25_12b.csv 的结果。

A1

	A	B	C	D	E	F	G	H	I
1		Chinese	English	Math	Natural	Society			
2	1	14	12	13	10	13			
3	2	13	14	11	10	15			
4	3	15	9	12	8	15			
5	4	15	10	13	10	15			
6	5	12	11	14	9	14			

out25_12a

	A	B	C	D	E	F	G	H	I
1	14	12	13	10	13				
2	13	14	11	10	15				
3	15	9	12	8	15				
4	15	10	13	10	15				
5	12	11	14	9	14				

out25_12b

25-4-2　读取 CSV 格式文件

Pandas 可以使用 read_csv( ) 读取 CSV 文件（也可以读取 TXT 文件），它的语法如下：

read_csv(path=None, sep=',', header=True, index_col=None, names=None,encoding=None, userows=None, usecols=None, …)

path：文件路径 (名称)。

sep：分隔字符，默认是‘,’。

header：设置那一行为字段标签，默认是 0。当参数有 names 时，此为 None。如果所读取的文件有字段标签时，就需设置此 header 值。

index_col：指出第几字段 column 是索引，默认是 None。

encoding：文件编码方式。

nrows：设置读取前几行。

usecols：设置读取那几字段。

程序实例 ch25_13.py：分别读取 ch25_12.py 所建立的 CSV 文件，然后打印。

```
1  # ch25_13.py
2  import pandas as pd
3
4  course = ['Chinese', 'English', 'Math', 'Natural', 'Society']
5  x = pd.read_csv("out25_12a.csv",index_col=0)
6  y = pd.read_csv("out25_12b.csv",names=course)
7  print(x)
8  print(y)
```

执行结果

```
==================== RESTART: D:/Python/ch25/ch25_13.py ====================
  Chinese  English  Math  Natural  Society
1      14       12    13       10       13
2      13       14    11       10       15
3      15        9    12        8       15
4      15       10    13       10       15
5      12       11    14        9       14
  Chinese  English  Math  Natural  Society
0      14       12    13       10       13
1      13       14    11       10       15
2      15        9    12        8       15
3      15       10    13       10       15
4      12       11    14        9       14
```

25-5　Pandas 绘图

Pandas 内有许多绘图函数，最常使用的是 plot( )，我们可以使用它为 Series 和 DataFrame 对象绘图。基本上这是 Pandas 模块将 matplotlib.pyplot 包装起来的一个绘图方法，所以程序设计时需要

import matplotlib.pyplot。这个 plot( ) 基本语法如下：

```
plot(x=None, y=None, kind="xx", title=None, legend=True,
rot=None, …)
```

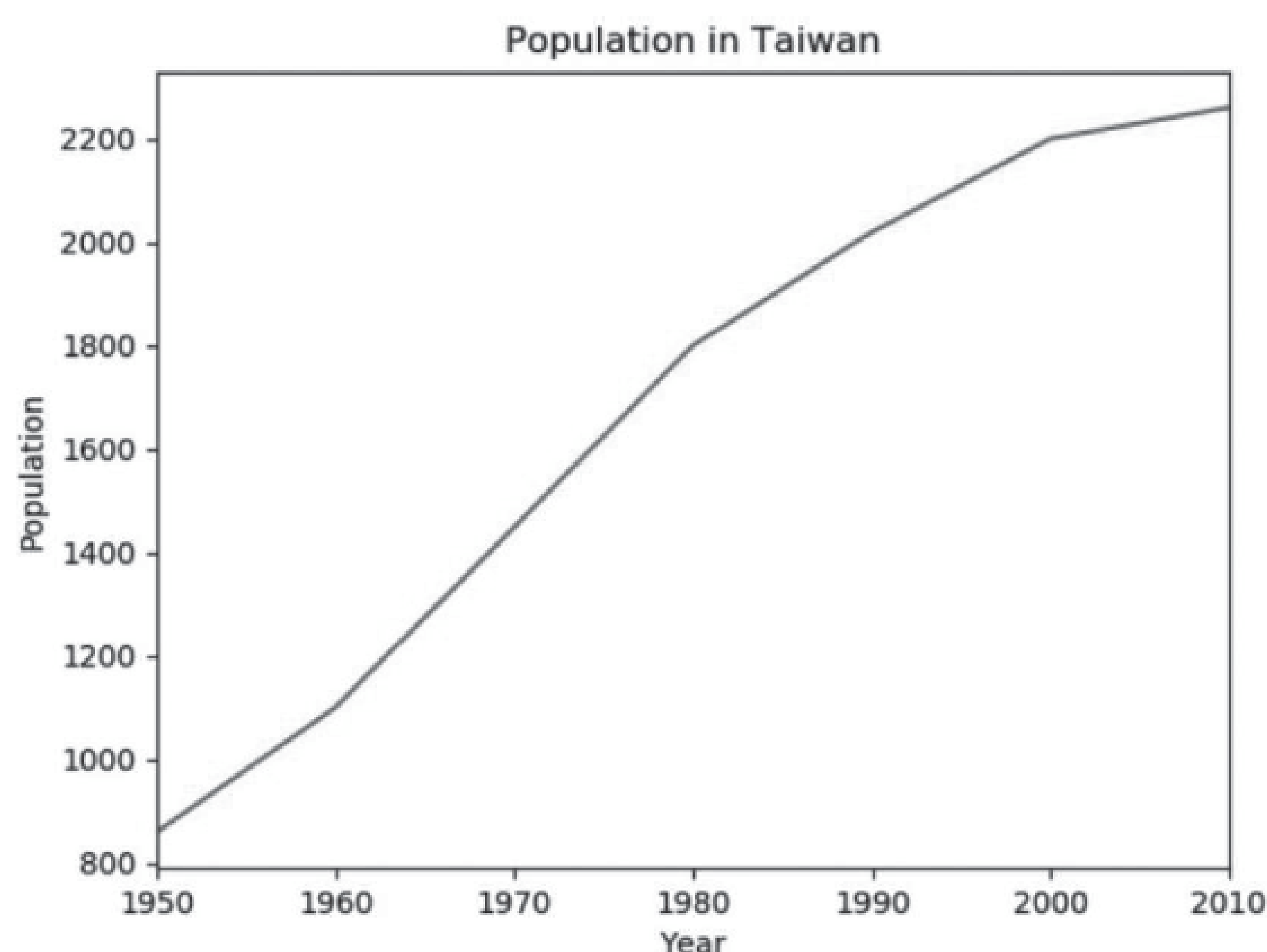
kind 是选择绘图模式，默认是 line，常见的选项有 bar、barh、hist、box、scatter 等。rot 是旋转刻度。

25-5-1 使用 Series 绘制折线图表

程序实例 ch25_14.py：建立一个 Series 对象 tw，这是纪录 1950—2010 年，每隔 10 年台湾人口的数据，单位是万人。

```
1  # ch25_14.py
2  import pandas as pd
3  import matplotlib.pyplot as plt
4
5  population = [860, 1100, 1450, 1800, 2020, 2200, 2260]
6  tw = pd.Series(population, index=range(1950, 2011, 10))
7  tw.plot(title='Population in Taiwan')
8  plt.xlabel("Year")
9  plt.ylabel("Population")
10 plt.show()
```

执行结果

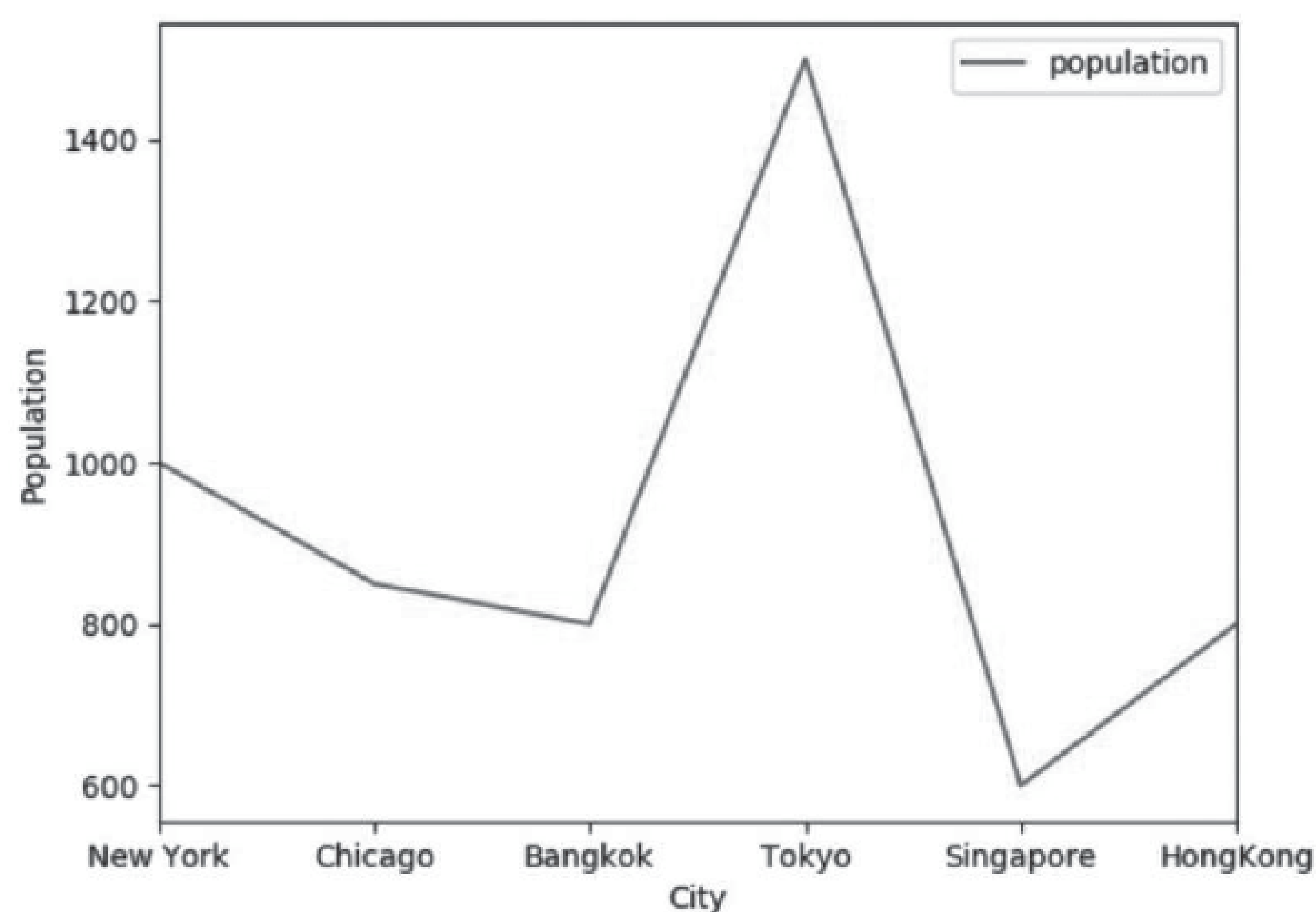


25-5-2 使用 DataFrame 绘制图表的基本知识

程序实例 ch25_15.py：设计一个世界大城市的人口图，制作 DataFrame 对象，然后绘制图表。

```
1  # ch25_15.py
2  import pandas as pd
3  import matplotlib.pyplot as plt
4
5  cities = {'population':[1000, 850, 800, 1500, 600, 800],
6            'town':['New York','Chicago','Bangkok','Tokyo',
7                    'Singapore','HongKong']}
8  tw = pd.DataFrame(cities, columns=['population'],index=cities['town'])
9
10 tw.plot(title='Population in the World')
11 plt.xlabel('City')
12 plt.ylabel("Population")
13 plt.show()
```

执行结果



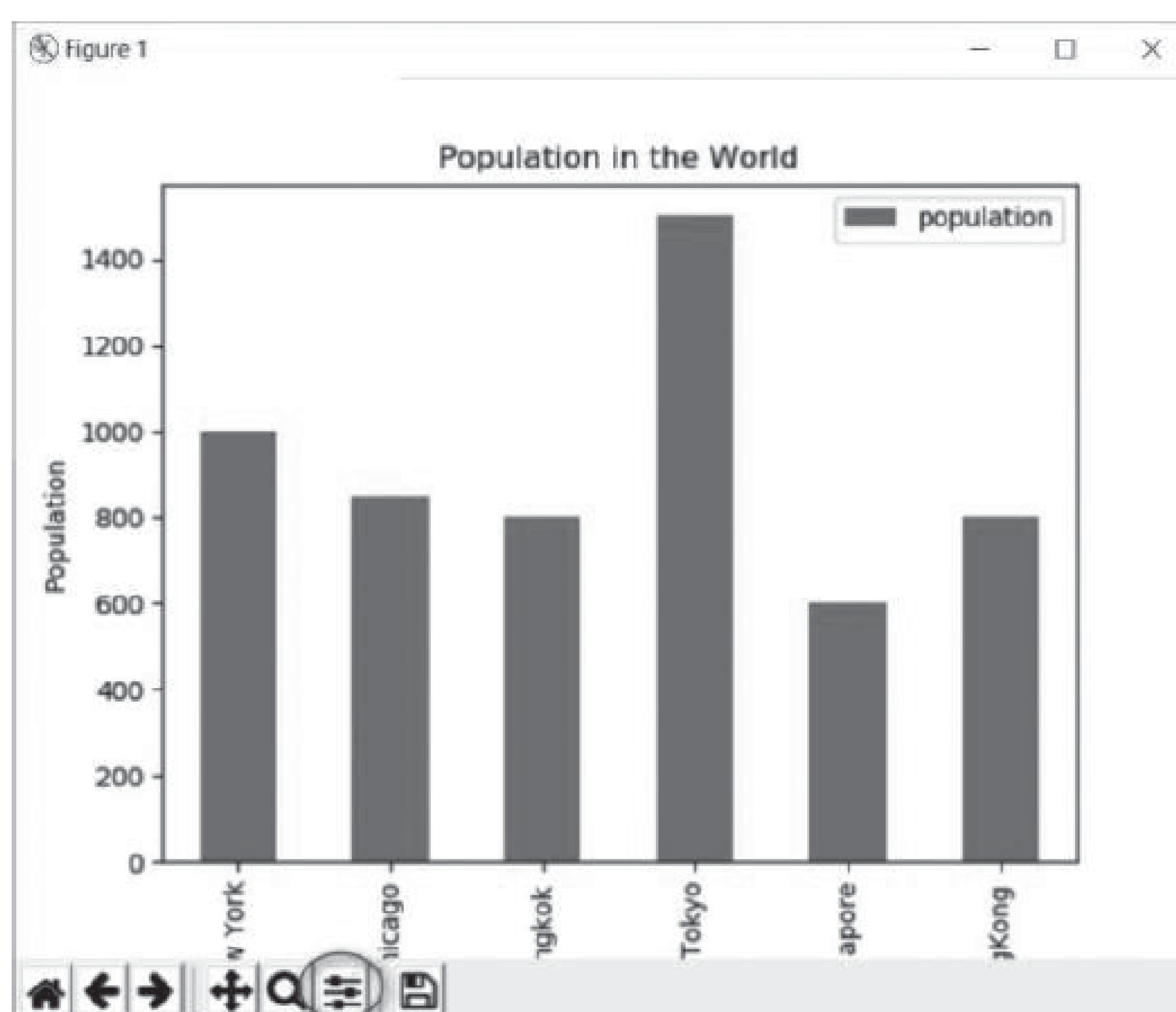
25-5-3　柱形图的设计

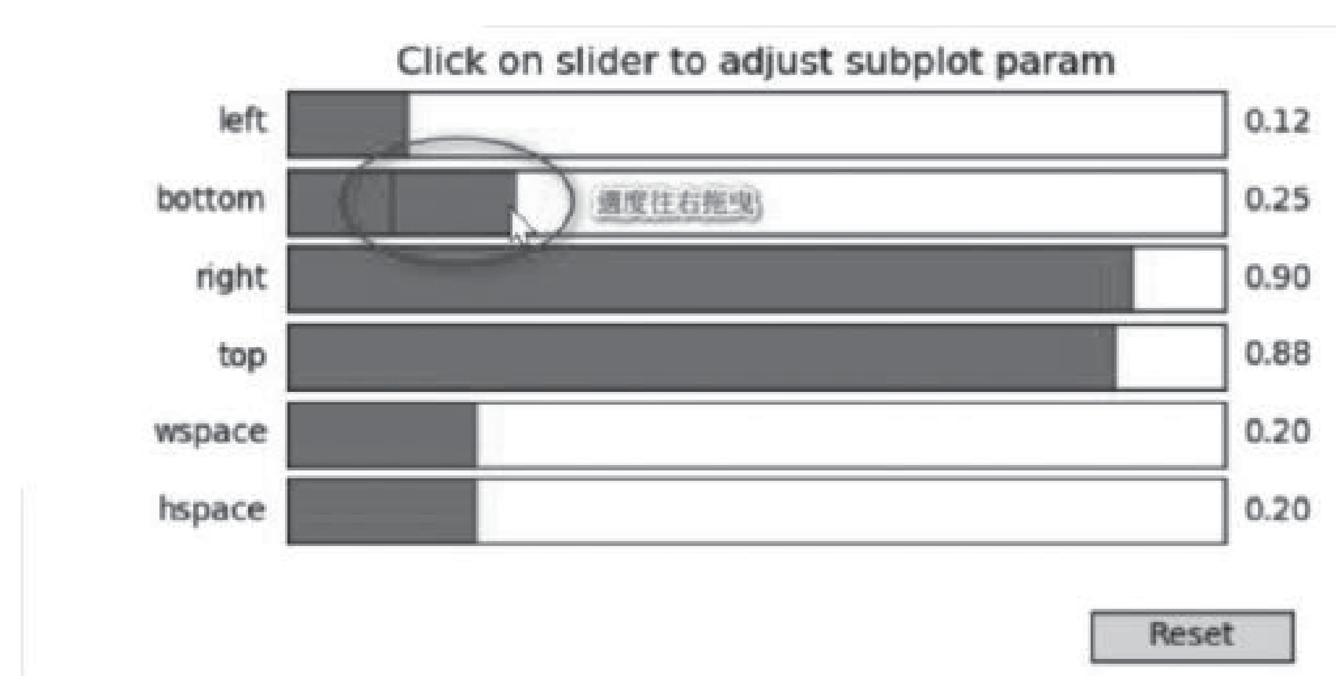
我们也可以使用适当的 kind 参数，更改不同的图表设计。

程序实例 ch25_16.py：使用柱形图，重新设计程序实例 ch25_15.py。

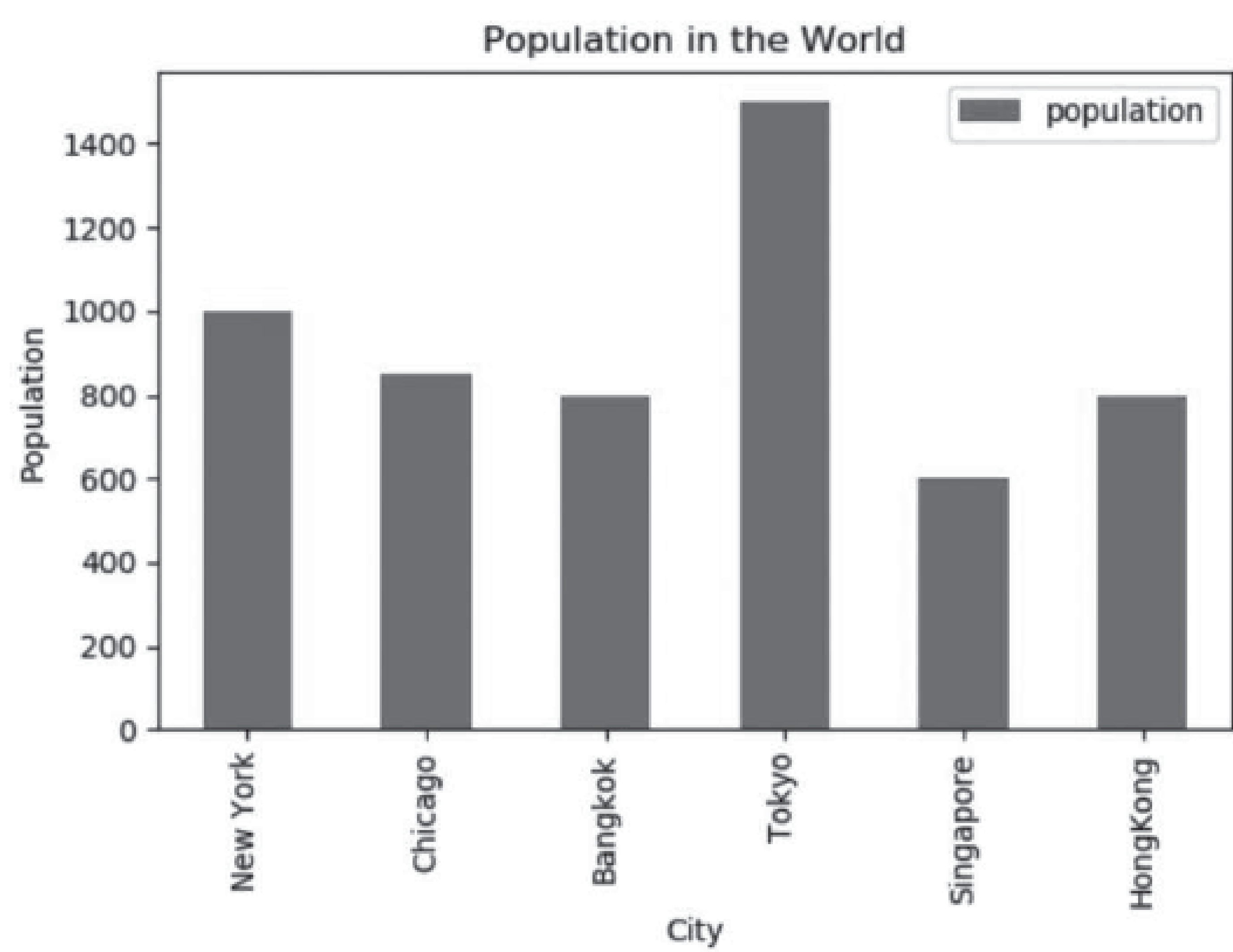
```
10  tw.plot(title='Population in the World',kind='bar')
```

执行结果 单击下方左图圈起的图示，再拖曳到 bottom 的位置。





原先 y 轴标签无法完全显示，现在可以了。



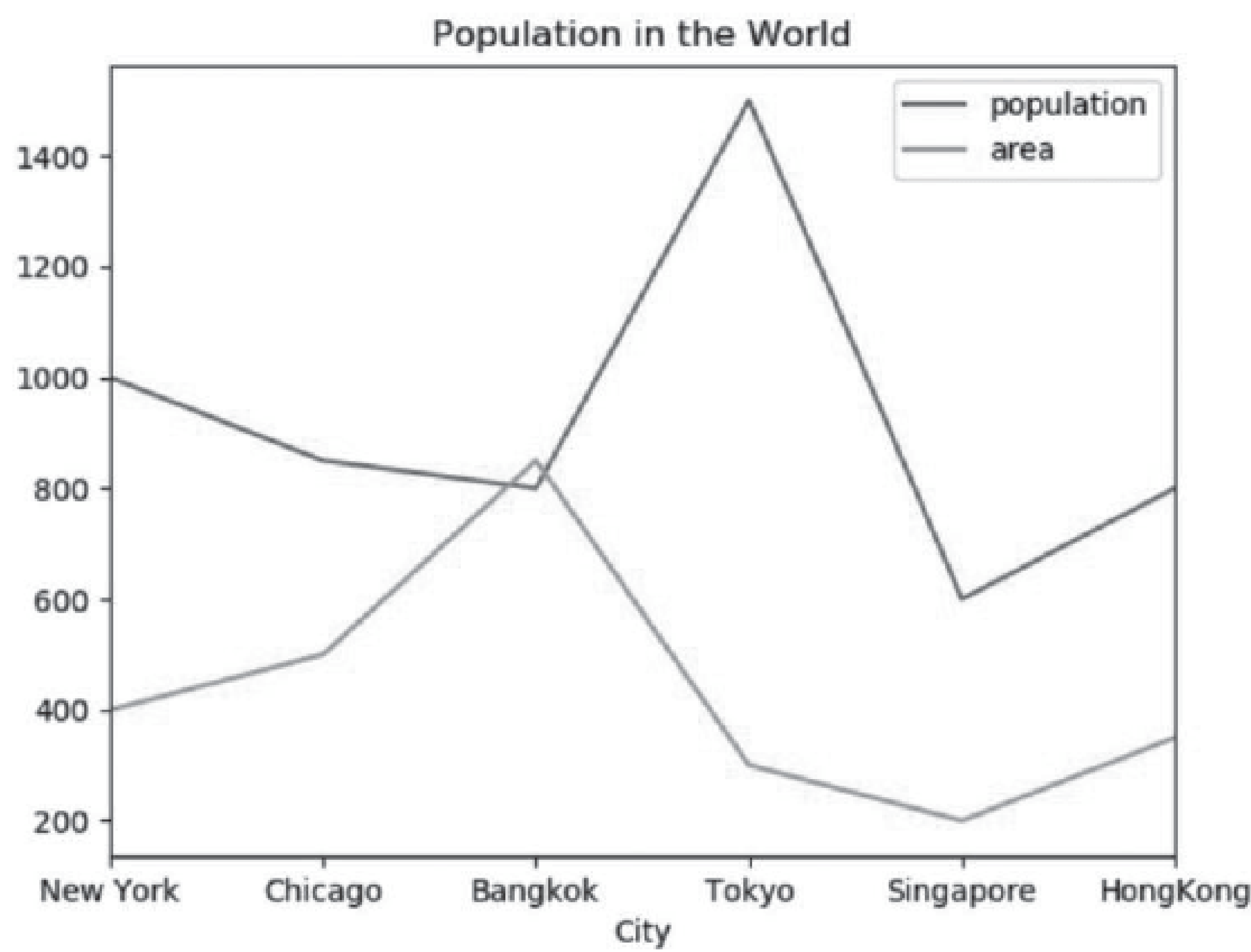
25-5-4 一个图表含不同数值数据

我们也可以使用一张图表建立多个数值数据，例如，下列是增加城市面积的数据实例。

程序实例 ch25_17.py：扩充 DataFrame，增加城市面积数据（平方千米）。

```
1  # ch25_17.py
2  import pandas as pd
3  import matplotlib.pyplot as plt
4
5  cities = {'population':[1000, 850, 800, 1500, 600, 800],
6            'area':[400, 500, 850, 300, 200, 320],
7            'town':['New York','Chicago','Bangkok','Tokyo',
8                    'Singapore','HongKong']}
9  tw = pd.DataFrame(cities, columns=['population','area'],index=cities['town'])
10
11 tw.plot(title='Population in the World')
12 plt.xlabel('City')
13 plt.show()
```

执行结果

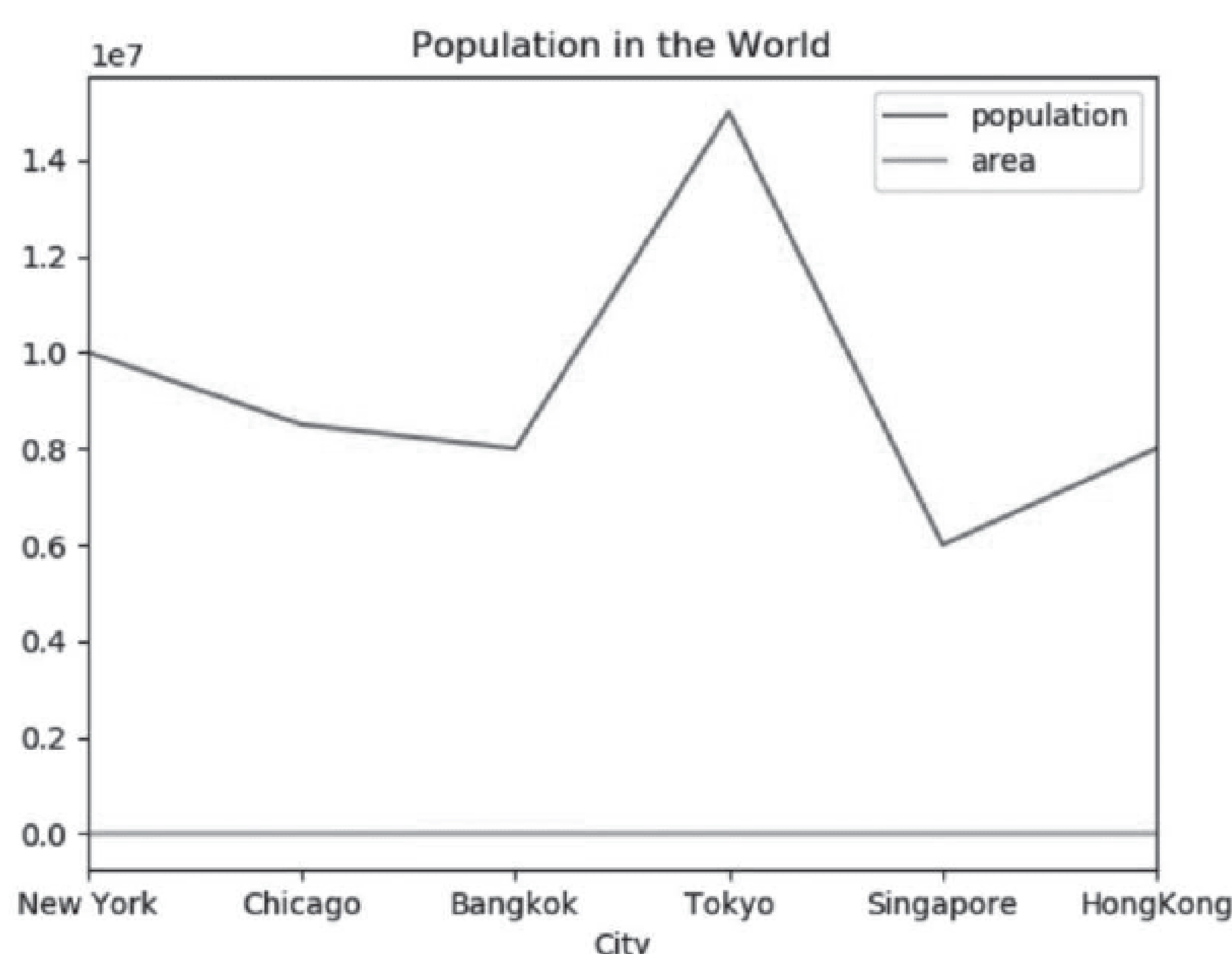


在上述程序设计中，笔者将人口数的单位设为“万人”，如果我们修改单位为“人”重新设计上述程序，则会因为面积与人口数相差太多，造成面积的数据无法正常显示。

程序实例 ch25_18.py：将人口单位改为“人”，重新设计 ch25_17.py。

```
1  # ch25_18.py
2  import pandas as pd
3  import matplotlib.pyplot as plt
4
5  cities = {'population':[10000000,8500000,8000000,15000000,6000000,8000000],
6            'area':[400, 500, 850, 300, 200, 320],
7            'town':['New York','Chicago','Bangkok','Tokyo',
8                    'Singapore','HongKong']}
9  tw = pd.DataFrame(cities, columns=['population','area'],index=cities['town'])
10
11 tw.plot(title='Population in the World')
12 plt.xlabel('City')
13 plt.show()
```

执行结果



若要解决这类问题，建议增加数值轴，具体做法可以参考下一小节。

25-5-5　多个数值轴的设计

使用 subplots( ) 可以在一个图表内显示多组不同轴的数据。程序第 11 行内容如下所示：

fig, ax = subplots( )　　　　# fig 是整体图表对象，ax 是第一个轴

第 16 行使用 twinx( ) 可以建立第 2 个数值轴，程序第 16 行内容如下：

ax2 = ax.twinx( )　　　　# 建立第 2 个轴对象 ax2

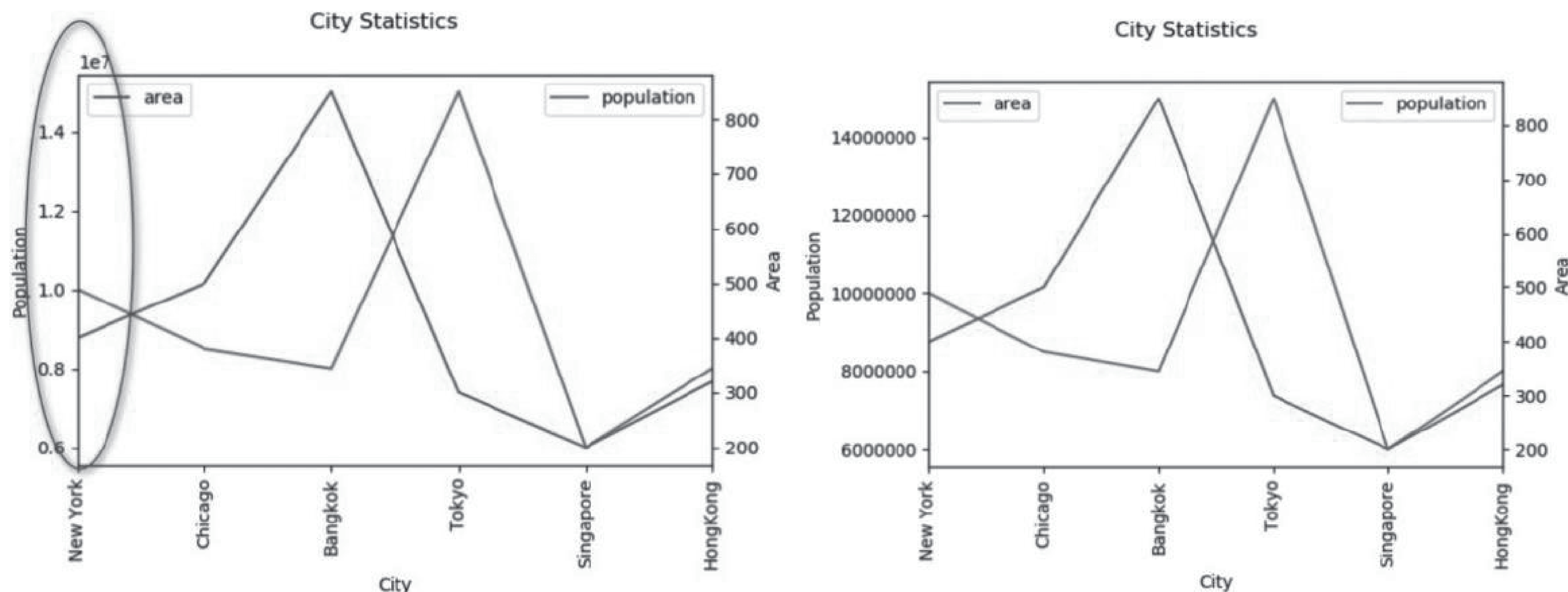
程序实例 ch25_19.py：用第 2 个轴的概念重新设计 ch25_18.py。

```
1  # ch25_19.py
2  import pandas as pd
3  import matplotlib.pyplot as plt
4
5  cities = {'population':[10000000,8500000,8000000,15000000,6000000,8000000],
6            'area':[400, 500, 850, 300, 200, 320],
7            'town':['New York','Chicago','Bangkok','Tokyo',
8                    'Singapore','HongKong']}
9  tw = pd.DataFrame(cities, columns=['population','area'],index=cities['town'])
10
11 fig, ax = plt.subplots()
12 fig.suptitle("City Statistics")
13 ax.set_ylabel("Population")
14 ax.set_xlabel("City")
15
```

```
16  ax2 = ax.twinx()
17  ax2.set_ylabel("Area")
18  tw['population'].plot(ax=ax,rot=90)        # 绘制人口数线
19  tw['area'].plot(ax=ax2, style='g-')        # 绘制面积线
20  ax.legend(loc=1)                           # 图例位置在右上
21  ax2.legend(loc=2)                          # 图例位置在左上
22  plt.show()
```

执行结果

下方左图是类似 ch25_16.py 调整的结果。



程序实例 ch25_20.py：重新设计 ch25_19.py，在左侧 y 轴不用科学计数法表示人口数，此例在第 15 行增加下列 ticklabel_format( )。

```
15  ax.ticklabel_format(style='plain')      # 不用科学记号表示
```

执行结果

可以参考上方右图。

以上概念也可以用于扩充第 3 个轴，只要设置 ax3 = ax.twinx( )，其余则参照新增轴的方法即可。

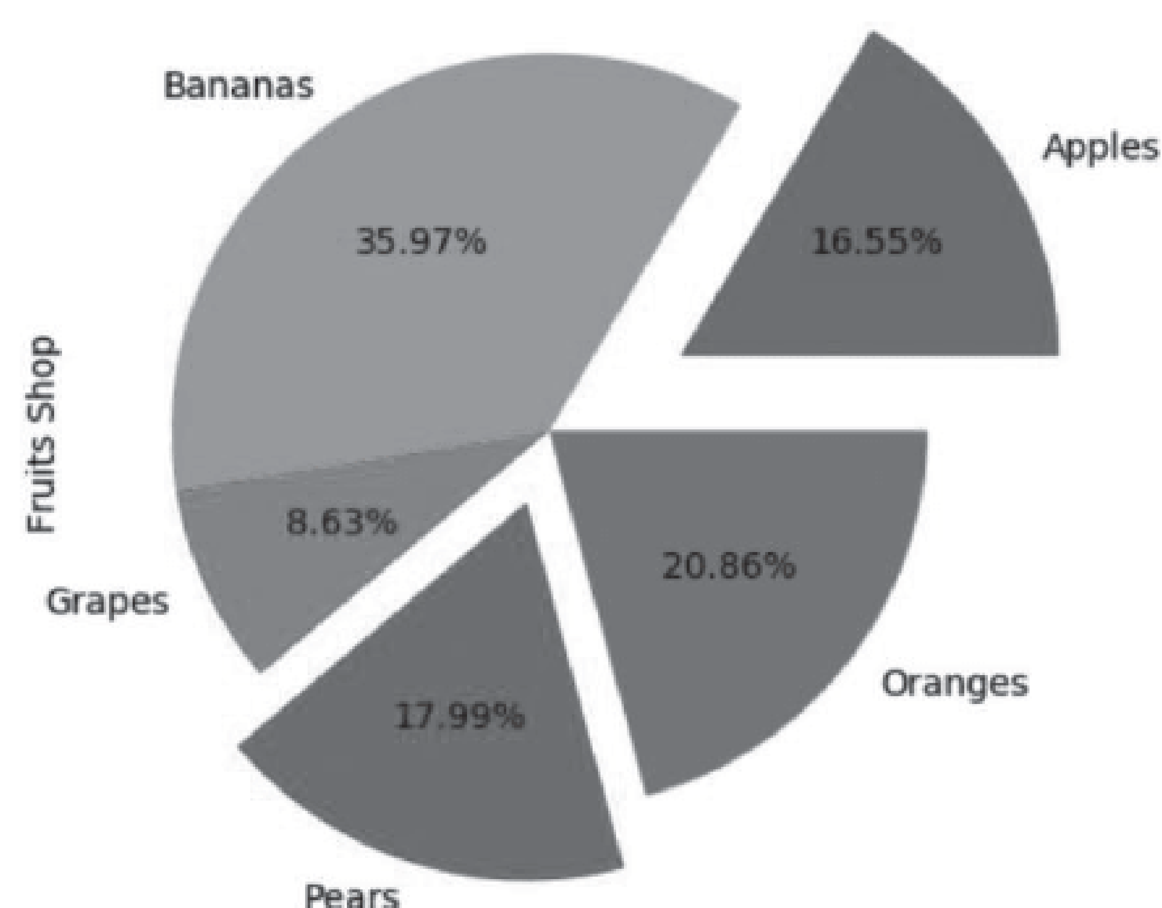
25-5-6 使用 Series 对象设计圆饼图

绘制圆饼图可以使用 plot.pie( )，有关 pie( ) 参数的知识可以参考 20-7 节。

程序实例 ch25_21.py：使用 Series 对象绘制圆饼图。

```
1   # ch25_21.py
2   import pandas as pd
3   import matplotlib.pyplot as plt
4
5   fruits = ['Apples', 'Bananas', 'Grapes', 'Pears', 'Oranges']
6   s = pd.Series([2300, 5000, 1200, 2500, 2900], index=fruits,
7                 name='Fruits Shop')
8   explode = [0.4, 0, 0, 0.2, 0]
9   s.plot.pie(explode = explode, autopct='%1.2f%%')
10  plt.show()
```

执行结果



25-6　时间序列（Time Series）

时间序列是指一系列的数据依时间次序列出。时间是指一系列的时间戳（timestamp），这些时间戳是相等间隔的时间点。音乐 mp3 文件或是一些声音文件，其实就是时间序列的应用，因为音频会依时间序列排成数据点，将这些数据点可视化，就可以组织成声音波形。这一节笔者先介绍 Python 的 datetime 模块，将它应用在 Series 对象建立时间序列，然后再介绍 Pandas 处理时间序列的工具。

25-6-1　时间模块 datetime

在 13-6 节笔者讲解过时间模块 time，这一节将讲解另一个时间模块 datetime，在使用前需要导入此模块：

```
from datetime import datetime
```

1. datetime 模块的数据形态 datetime

datetime 模块内有一个数据形态 datetime，可以用它代表一个特定时间，有一个 now( ) 方法可以列出现在时间。

程序实例 ch25_21_1.py：列出现在时间。

```
1  # ch25_21_1.py
2  from datetime import datetime
3
4  timeNow = datetime.now()
5  print(type(timeNow))
6  print("现在时间 : ", timeNow)
```

执行结果

```
==================== RESTART: D:\Python\ch25\ch25_21_1.py ====================
<class 'datetime.datetime'>
现在时间 :  2019-06-12 16:39:19.732434
```

我们也可以使用属性 year、month、day、hour、minute、second、microsecond（百万分之一秒），

获得上述时间的个别内容。

程序实例 ch25_21_2.py：列出时间的个别内容。

```
1  # ch25_21_2.py
2  from datetime import datetime
3
4  timeNow = datetime.now()
5  print(type(timeNow))
6  print("现在时间 : ", timeNow)
7  print("年 : ", timeNow.year)
8  print("月 : ", timeNow.month)
9  print("日 : ", timeNow.day)
10 print("时 : ", timeNow.hour)
11 print("分 : ", timeNow.minute)
12 print("秒 : ", timeNow.second)
```

执行结果

```
==================== RESTART: D:\Python\ch25\ch25_21_2.py ====================
<class 'datetime.datetime'>
现在时间 :  2019-06-12 16:41:20.104197
年 :  2019
月 :  6
日 :  12
时 :  16
分 :  41
秒 :  20
```

另一个属性百万分之一秒 microsecond，一般在程序中比较少用。

2. 设置特定时间

当你了解了获得现在时间的方式后，其实可以用下列方法设置一个特定时间：

xtime = datetime.datetime(年，月，日，时，分，秒)

上述 xtime 就是一个特定时间。

程序实例 ch25_21_3.py：设置程序循环执行到 2019 年 3 月 11 日 22 点 271 分 0 秒将苏醒停止打印 program is sleeping，然后打印 Wake up。

```
1  # ch25_21_3.py
2  from datetime import datetime
3
4  timeStop = datetime(2019,3,11,22,27,10)
5  while datetime.now() < timeStop:
6      print("Program is sleeping.", end="")
7  print("Wake up")
```

执行结果

```
==================== RESTART: D:/Python/ch25/ch25_21_3.py ====================
Program is sleeping.Program is sleeping.Program is sleeping.Program is sleeping.
Program is sleeping.Program is sleeping.Program is sleeping.Program is sleeping.
Program is sleeping.Program is sleeping.Program is sleeping.Program is sleeping.
Program is sleeping.Program is sleeping.Program is sleeping.Program is sleeping.
Program is sleeping.Program is sleeping.Program is sleeping.Program is sleeping.
Program is sleeping.Program is sleeping.Program is sleeping.Program is sleeping.
Program is sleeping.Program is sleeping.Program is sleeping.Program is sleeping.
Program is sleeping.Program is sleeping.Program is sleeping.Program is sleeping.
Program is sleeping.Program is sleeping.Program is sleeping.Program is sleeping.
Program is sleeping.Program is sleeping.Program is sleeping.Program is sleeping.
Program is sleeping.Program is sleeping.Program is sleeping.Program is sleeping.
Program is sleeping.Program is sleeping.Program is sleeping.Program is sleeping.
Program is sleeping.Program is sleeping.Program is sleeping.Program is sleeping.
Program is sleeping.Program is sleeping.Program is sleeping.Program is sleeping.
Program is sleeping.Program is sleeping.Program is sleeping.Program is sleeping.
Program is sleeping.Program is sleeping.Program is sleeping.Program is sleeping.
Program is sleeping.Program is sleeping.Program is sleeping.Program is sleeping.
Program is sleeping.Program is sleeping.Program is sleeping.Program is sleeping.
Program is sleeping.Program is sleeping.Program is sleeping.Program is sleeping.
Program is sleeping.Wake up
```

3. 一段时间 timedelta

这是 datetime 的数据类型，代表的是一段时间，可以用下列方式指定一段时间。

```
deltaTime=datetime.timedelta(weeks=xx,days=xx,hours=xx,minutes=xx,
seocnds=xx)
```

上述 xx 代表设置的单位数。

一段时间的对象只有 3 个属性，days 代表日数、seconds 代表秒数、microseconds 代表百万分之一秒。

程序实例 ch25_21_4.py：打印一段时间的日数、秒数和百万分之几秒。

```
1  # ch25_21_4.py
2  from datetime import datetime, timedelta
3
4  deltaTime = timedelta(days=3,hours=5,minutes=8,seconds=10)
5  print(deltaTime.days, deltaTime.seconds, deltaTime.microseconds)
```

执行结果

```
==================== RESTART: D:/Python/ch25/ch25_21_4.py ====================
3 18490 0
```

上述 5 小时 8 分 10 秒被总计为 18940 秒。有一个方法 total_second( ) 可以将一段时间转成秒数。

程序实例 ch25_21_5.py：重新设计 ch25_21_4.py，将一段时间转成秒数。

```
1  # ch25_21_5.py
2  from datetime import datetime, timedelta
3
4  deltaTime = timedelta(days=3,hours=5,minutes=8,seconds=10)
5  print(deltaTime.total_seconds())
```

执行结果

```
==================== RESTART: D:/Python/ch25/ch25_21_5.py ====================
277690.0
```

25-6-2　使用 Python 的 datetime 模块建立含时间戳的 Series 对象

对于时间序列（Time Series）而言，基本上就是将索引（index）用日期取代。

程序实例 ch25_21_6.py：使用 datetime 建立含 5 天的 Series 对象和打印，这 5 天数据则是使用列表 [34, 44, 65, 53, 39]，同时列出时间序列对象的数据类型。

```
1  # ch25_21_6.py
2  import pandas as pd
3  from datetime import datetime, timedelta
4
5  ndays = 5
6  start = datetime(2019, 3, 11)
7  dates = [start + timedelta(days=x) for x in range(0, ndays)]
8  data = [34, 44, 65, 53, 39]
9  ts = pd.Series(data, index=dates)
10 print(type(ts))
11 print(ts)
```

执行结果

```
==================== RESTART: D:/Python/ch25/ch25_21_6.py ====================
<class 'pandas.core.series.Series'>
2019-03-11    34
2019-03-12    44
2019-03-13    65
2019-03-14    53
2019-03-15    39
dtype: int64
```

我们也可以使用 ts.index 列出此时间序列的索引，以了解 Series 的索引结构。

```
>>> ts.index
DatetimeIndex(['2019-03-11', '2019-03-12', '2019-03-13', '2019-03-14',
               '2019-03-15'],
              dtype='datetime64[ns]', freq=None)
```

时间序列是允许相同索引执行加法或代数运算的。

程序实例 ch25_21_7.py：扩充前一个程序建立相同时间戳的 Series 对象，然后计算两个 Series 对象的相加与计算平均。

```
1  # ch25_21_7.py
2  import pandas as pd
3  from datetime import datetime, timedelta
4
5  ndays = 5
6  start = datetime(2019, 3, 11)
7  dates = [start + timedelta(days=x) for x in range(0, ndays)]
8  data1 = [34, 44, 65, 53, 39]
9  ts1 = pd.Series(data1, index=dates)
10
11 data2 = [34, 44, 65, 53, 39]
12 ts2 = pd.Series(data2, index=dates)
13
14 addts = ts1 + ts2
15 print("ts1+ts2")
16 print(addts)
17
18 meants = (ts1 + ts2)/2
19 print("(ts1+ts2)/2")
20 print(meants)
```

执行结果

```
==================== RESTART: D:/Python/ch25/ch25_21_7.py ====================
ts1+ts2
2019-03-11     68
2019-03-12     88
2019-03-13    130
2019-03-14    106
2019-03-15     78
dtype: int64
(ts1+ts2)/2
2019-03-11    34.0
2019-03-12    44.0
2019-03-13    65.0
2019-03-14    53.0
2019-03-15    39.0
dtype: float64
```

在上述 ch25_21_7.py 的计算过程中，如果时间戳不一样，将产生 NaN 数值。

程序实例 ch25_21_8.py：重新设计前一个程序，执行两个 Series 对象相加，但是部分时间戳是不同。

```
1  # ch25_21_8.py
2  import pandas as pd
3  from datetime import datetime, timedelta
4
5  ndays = 5
6  start = datetime(2019, 3, 11)
7  dates1 = [start + timedelta(days=x) for x in range(0, ndays)]
8  data1 = [34, 44, 65, 53, 39]
9  ts1 = pd.Series(data1, index=dates1)
10
11 dates2 = [start - timedelta(days=x) for x in range(0, ndays)]
12 data2 = [34, 44, 65, 53, 39]
13 ts2 = pd.Series(data2, index=dates2)
14
15 addts = ts1 + ts2
16 print("ts1+ts2")
17 print(addts)
```

执行结果

```
=================== RESTART: D:/Python/ch25/ch25_21_8.py ===================
ts1+ts2
2019-03-07     NaN
2019-03-08     NaN
2019-03-09     NaN
2019-03-10     NaN
2019-03-11    68.0
2019-03-12     NaN
2019-03-13     NaN
2019-03-14     NaN
2019-03-15     NaN
dtype: float64
```

25-6-3　Pandas 的时间区间方法

Pandas 的 date_range( ) 可以产生时间区间，我们可以更方便地将此方法应用在前一小节的程序中。

程序实例 ch25_21_9.py：使用 date_range( ) 重新设计 ch25_21_6.py。

```
1  # ch25_21_9.py
2  import pandas as pd
3
4  dates = pd.date_range('3/11/2019', '3/15/2019')
5  data = [34, 44, 65, 53, 39]
6  ts = pd.Series(data, index=dates)
7  print(type(ts))
8  print(ts)
```

执行结果

```
==================== RESTART: D:/Python/ch25/ch25_21_9.py ====================
<class 'pandas.core.series.Series'>
2019-03-11    34
2019-03-12    44
2019-03-13    65
2019-03-14    53
2019-03-15    39
Freq: D, dtype: int64
```

结果基本上与 ch25_21_6.py 相同，但是多了 Freq: D，表示索引是日期。如果这时我们输入 ts.index 也将获得一样的结果。

```
>>> ts.index
DatetimeIndex(['2019-03-11', '2019-03-12', '2019-03-13', '2019-03-14',
               '2019-03-15'],
              dtype='datetime64[ns]', freq='D')
```

上述我们使用 date_range( ) 方法时，是放了起始日期与终止日期，我们也可以用起始日期（start=）再加上期间（periods=），或是终止日期（end=）再加上期间（periods=）设置时间戳。

实例 1：使用起始日期，加上期间设置时间索引。

```
>>> dates = pd.date_range(start='2019-03-11', periods=5)
>>> dates
DatetimeIndex(['2019-03-11', '2019-03-12', '2019-03-13', '2019-03-14',
               '2019-03-15'],
              dtype='datetime64[ns]', freq='D')
```

实例 2：使用终止日期，加上期间设置时间索引。

```
>>> dates = pd.date_range(end='2019-03-15', periods=5)
>>> dates
DatetimeIndex(['2019-03-11', '2019-03-12', '2019-03-13', '2019-03-14',
               '2019-03-15'],
              dtype='datetime64[ns]', freq='D')
```

此外在设置 data_range( ) 时，如果设置参数 freq='B'，可以让时间索引只包含工作日（work day），相当于假日（周六与周日）不包含在时间索引内。

实例 3：设置 2019 年 3 月 1 日—3 月 7 日的时间索引，设置参数 freq='B' 并观察执行结果。

```
>>> dates = pd.date_range('2019-03-01', '2019-03-07', freq='B')
>>> dates
DatetimeIndex(['2019-03-01', '2019-03-04', '2019-03-05', '2019-03-06',
               '2019-03-07'],
              dtype='datetime64[ns]', freq='B')
```

由于 3 月 2 日是周六，3 月 3 日是周日，所以最后皆不在时间索引内。若是设置 freq='M'，代表时间索引是两个时间点之间的月底。

实例 4：观察 freq='M' 的执行结果。

```
>>> dates = pd.date_range('2020-01-05', '2020-04-08', freq='M')
>>> dates
DatetimeIndex(['2020-01-31', '2020-02-29', '2020-03-31'], dtype='datetime64[ns]',
freq='M')
```

也可以使用 freq=W-Mon，Mon 是周一的缩写，两个时间点之间，代表每周一皆是时间索引，可以应用在其他日。

实例 5：观察 freq='W-Mon' 的执行结果。

```
>>> dates = pd.date_range('2019-03-01', '2019-03-31', freq='W-Mon')
>>> dates
DatetimeIndex(['2019-03-04', '2019-03-11', '2019-03-18', '2019-03-25'], dtype='d
atetime64[ns]', freq='W-MON')
```

其他常见的 freq 设置如下：

A：年末

AS：年初

Q：季末

QS：季初

H：小时

T：分钟

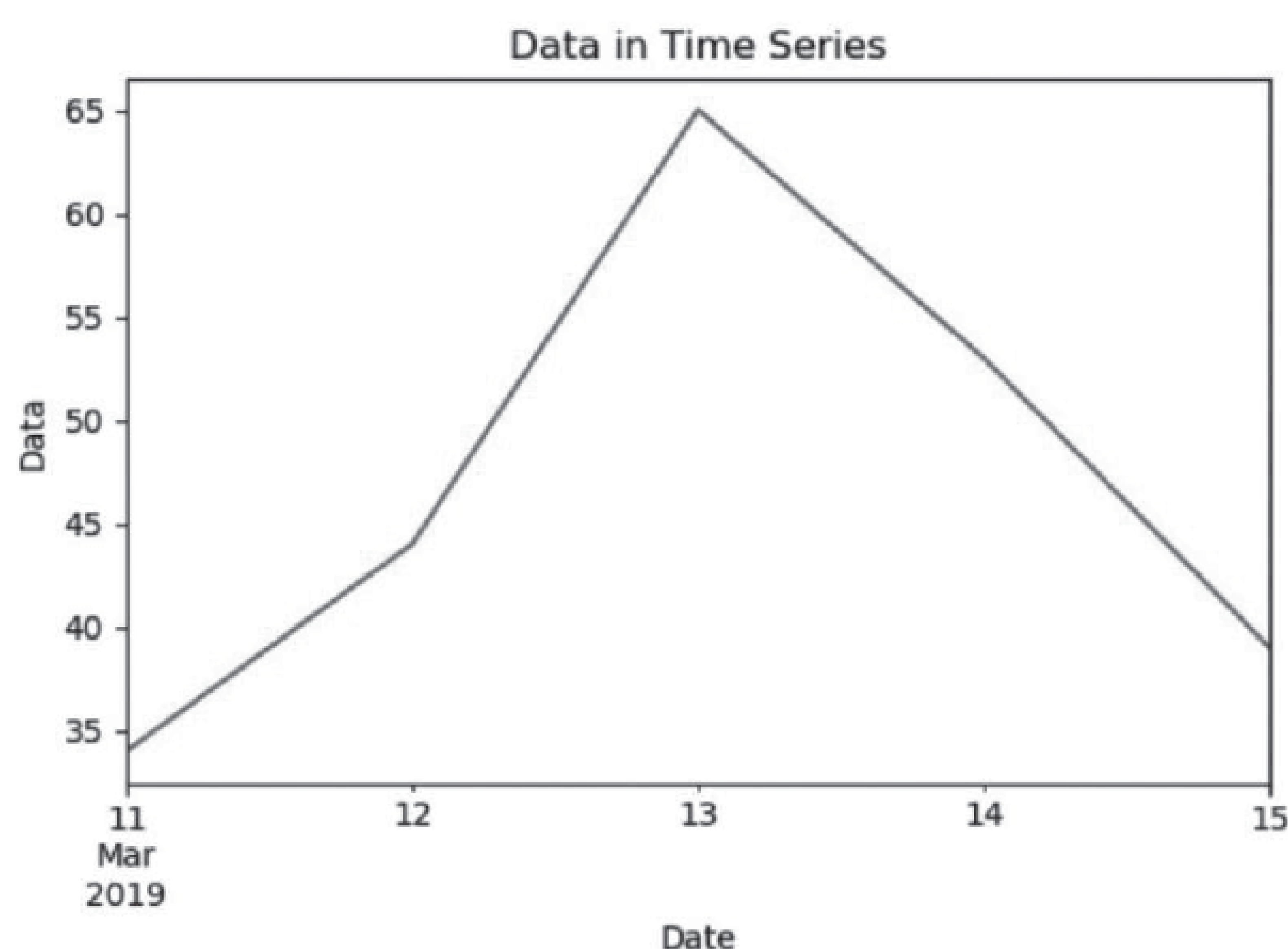
S：秒

25-6-4　将时间序列绘制折线图

实例 ch25_21_10.py：将 ch25_21_9.py 的时间序列绘制折线图。

```
1  # ch25_21_10.py
2  import pandas as pd
3  import matplotlib.pyplot as plt
4
5  dates = pd.date_range('3/11/2019', '3/15/2019')
6  data = [34, 44, 65, 53, 39]
7  ts = pd.Series(data, index=dates)
8  ts.plot(title='Data in Time Series')
9  plt.xlabel("Date")
10 plt.ylabel("Data")
11 plt.show()
```

执行结果

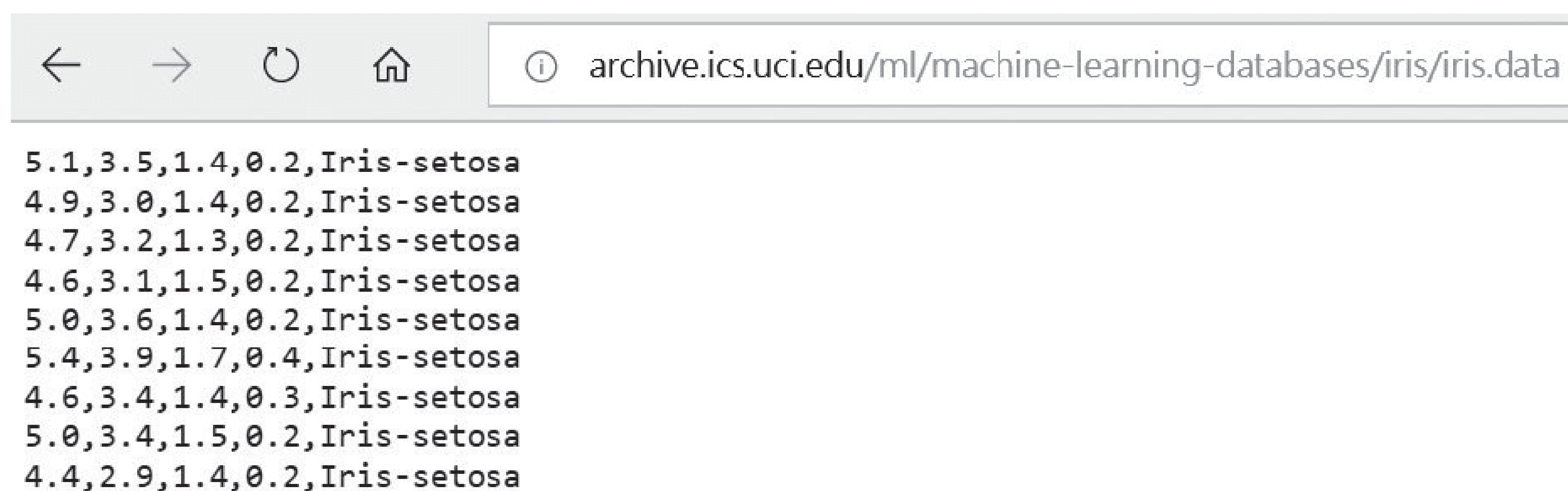


25-7　专题——鸢尾花

在数据分析领域有一组很有名的数据集 iris.csv，这是加州大学尔湾分校机器学习中常被应用的数据，这些数据是由美国植物学家艾德加安德森（Edgar Anderson）在加拿大 Gaspesie 半岛实际测量鸢尾花所采集的，读者可以由下列网页了解此数据集：

http://archive.ics.uci.edu/ml/machine-learning-databases/iris/

进入后将看到下列部分内容：




```
5.1,3.5,1.4,0.2,Iris-setosa
4.9,3.0,1.4,0.2,Iris-setosa
4.7,3.2,1.3,0.2,Iris-setosa
4.6,3.1,1.5,0.2,Iris-setosa
5.0,3.6,1.4,0.2,Iris-setosa
5.4,3.9,1.7,0.4,Iris-setosa
4.6,3.4,1.4,0.3,Iris-setosa
5.0,3.4,1.5,0.2,Iris-setosa
4.4,2.9,1.4,0.2,Iris-setosa
```

总共有 150 笔数据，在这个数据集中总共有 5 个字段，从左到右分别代表的意义如下：

花萼长度（sepal length）

花萼宽度（sepal width）

花瓣长度（petal length）

花瓣宽度（petal width）

鸢尾花类别（species, 有 setosa、versicolor、virginica）

这一个专题中，笔者将教导读者使用 Python 网络爬虫功能下载、存储成 iris.txt 与 iris.csv，然后一步一步使用此数据并配合 Pandas 功能执行分析。

25-7-1　网络爬虫

其实网络爬虫的知识可以用整本书做解说，此小节笔者将解说最基本的部分。所谓的网络爬虫其实就是下载网页信息，甚至可以说下载网页的 HTML 文件，在 Python 可以使用模块 requests，使用下列指令下载此模块：

```
pip install requests
```

在这个模块内，可以使用 request.get(url) 取得指定网址（url）的 HTML 文件，由鸢尾花资料及网页可以发现此网页内容非常单纯，没有其他 HTML 标签，所以可以直接读取然后储存。

程序实例 ch25_22.py：读取加州大学鸢尾花数据集网页，然后将此数据集储存成 iris.csv。

```
1  # ch25_22.py
2  import requests
3
4  url = 'http://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data'
5  try:
6      htmlfile = requests.get(url)                          # 将文件下载至htmlfile
7      print('下载成功')
8  except Exception as err:
9      print('下载失败')
10
11 fn = 'iris.csv'                                           # 未来存储鸢尾花的文件
12 with open(fn, 'wb') as fileobj:                           # 打开iris.csv
13     for diskstorage in htmlfile.iter_content(10240):
14         size = fileobj.write(diskstorage)                 # 写入
```

执行结果

```
==================== RESTART: D:\Python\ch25\ch25_22.py ====================
下载成功
```

这时在 ch25 文件夹可以看到 iris.csv，打开后可以得到下列结果。

A1　　5.1

	A	B	C	D	E	F	G	H	I
1	5.1	3.5	1.4	0.2	Iris-setosa				
2	4.9	3	1.4	0.2	Iris-setosa				
3	4.7	3.2	1.3	0.2	Iris-setosa				
4	4.6	3.1	1.5	0.2	Iris-setosa				
5	5	3.6	1.4	0.2	Iris-setosa				
6	5.4	3.9	1.7	0.4	Iris-setosa				
7	4.6	3.4	1.4	0.3	Iris-setosa				
8	5	3.4	1.5	0.2	Iris-setosa				

iris

上述程序的第 13 行中，笔者用 for 循环一次写入 10240 字节的数据，直到全部写入完成。

25-7-2　将鸢尾花数据集转成 DataFrame

程序实例 ch25_23.py：读取 iris.csv，为此数据集加上域名，然后列出此数据集的长度和内容。

```
1  # ch25_23.py
2  import pandas as pd
3
4  colName = ['sepal_len','sepal_wd','petal_len','petal_wd','species']
5  iris = pd.read_csv('iris.csv', names = colName)
6  print('数据集长度 : ', len(iris))
7  print(iris)
```

执行结果



建立好上述 DataFrame 后，也可以使用 describe( ) 获得数据的数量、均值、标准偏差值、最小值、最大值、各分位数的值。

实例：使用 describe( ) 列出 iris 的相关数据。

```
>>> iris.describe()
        sepal_len    sepal_wd   petal_len    petal_wd
count  150.000000  150.000000  150.000000  150.000000
mean     5.843333    3.054000    3.758667    1.198667
std      0.828066    0.433594    1.764420    0.763161
min      4.300000    2.000000    1.000000    0.100000
25%      5.100000    2.800000    1.600000    0.300000
50%      5.800000    3.000000    4.350000    1.300000
75%      6.400000    3.300000    5.100000    1.800000
max      7.900000    4.400000    6.900000    2.500000
```

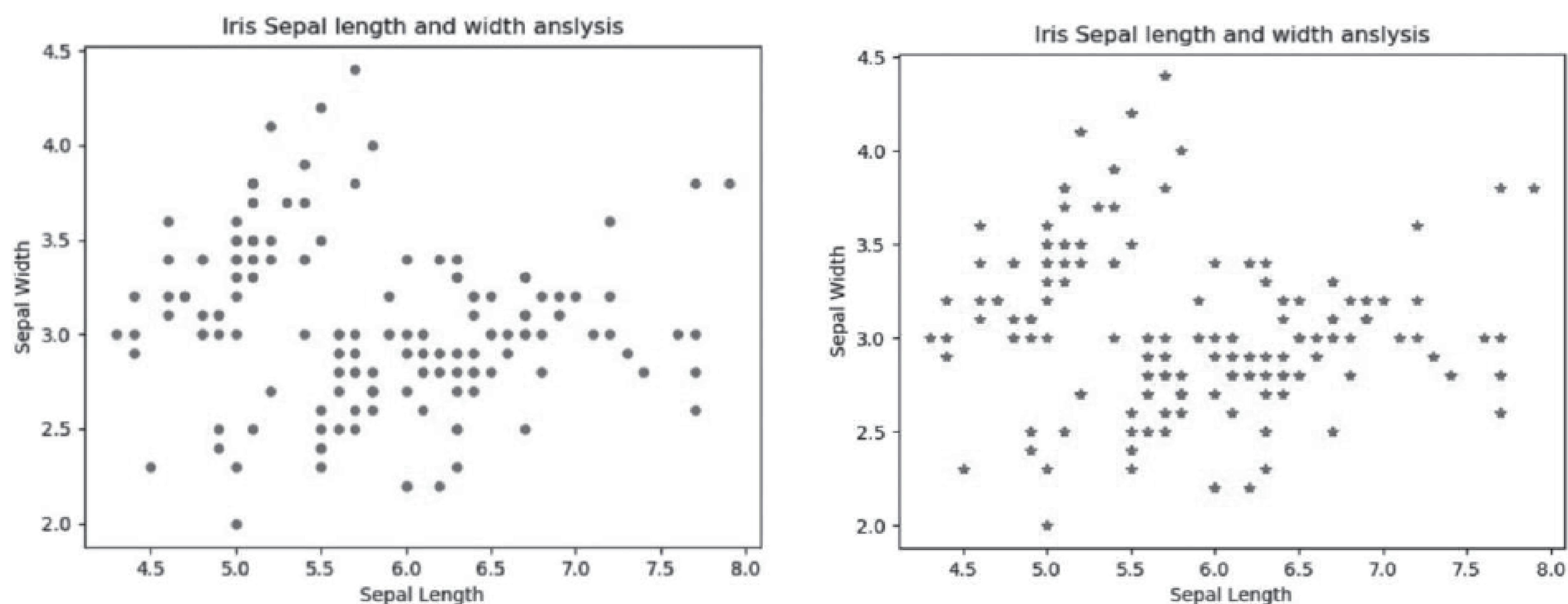
25-7-3　散点图的制作

绘制散点图可以使用 plot(…..,kind='scatter')，另外还要给予 x 轴和 y 轴坐标数组，由于是由 DataFrame 呼叫 plot( )，所以可以直接使用字段 column 名称即可。

程序实例 ch25_24.py：绘制（Sepal Length, Sepal Width）之散点图。

```
 1  # ch25_24.py
 2  import pandas as pd
 3  import matplotlib.pyplot as plt
 4
 5  colName = ['sepal_len','sepal_wd','petal_len','petal_wd','species']
 6  iris = pd.read_csv('iris.csv', names = colName)
 7
 8  iris.plot(x='sepal_len',y='sepal_wd',kind='scatter')
 9  plt.xlabel('Sepal Length')
10  plt.ylabel('Sepal Width')
11  plt.title('Iris Sepal length and width anslysis')
12  plt.show()
```

执行结果



其实绘制这类图表，也可以用绘点 plot( ) 方式完成。

程序实例 ch25_25.py：使用 plot( ) 方式完成，笔者尝试用不同颜色和点标记，这个程序只修改下列内容：

```
8  plt.plot(iris['sepal_len'],iris['sepal_wd'],'*',color='g')
```

执行结果 可参考上方右图。

对于这类数据分析而言，如果想要了解各品种鸢尾花的花萼长度与宽度之间的关系，需要将鸢尾花数据集依据品种（species）先分离，然后将不同品种的鸢尾花数据绘制在同一图表内，这样就可以一目了然。下列是将不同品种鸢尾花数据提取出来的方法。

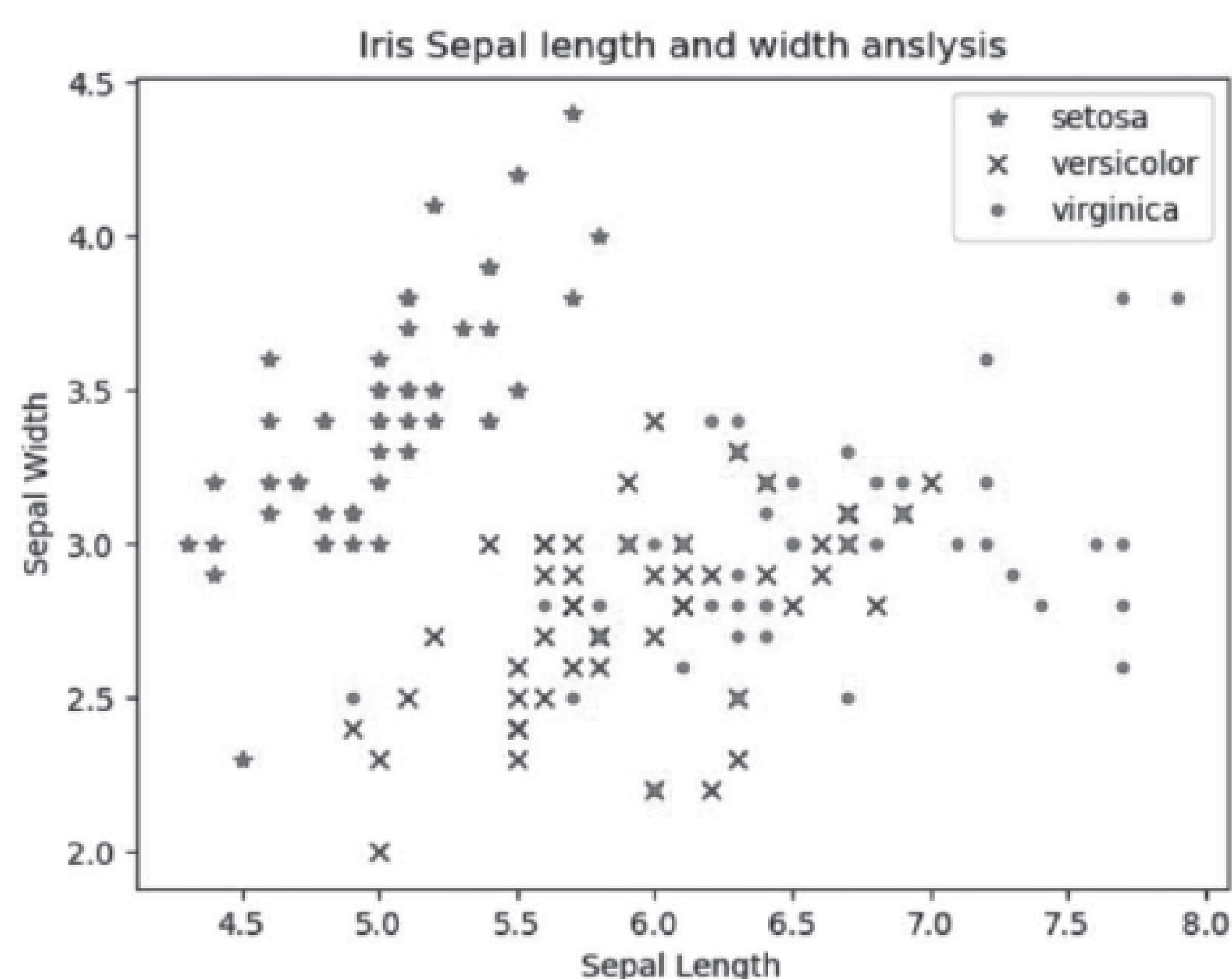
实例：延续先前实例，提取品种是 versicolor 的鸢尾花。

```
>>> iris_versicolor = iris[iris['species'] == 'Iris-versicolor']
>>> print(iris_versicolor)
    sepal_len  sepal_wd  petal_len  petal_wd          species
50        7.0       3.2        4.7       1.4  Iris-versicolor
51        6.4       3.2        4.5       1.5  Iris-versicolor
52        6.9       3.1        4.9       1.5  Iris-versicolor
53        5.5       2.3        4.0       1.3  Iris-versicolor
54        6.5       2.8        4.6       1.5  Iris-versicolor
55        5.7       2.8        4.5       1.3  Iris-versicolor
```

程序实例 ch25_26.py：将不同的鸢尾花的花萼使用不同的标记绘制散点图。

```
1  # ch25_26.py
2  import pandas as pd
3  import matplotlib.pyplot as plt
4
5  colName = ['sepal_len','sepal_wd','petal_len','petal_wd','species']
6  iris = pd.read_csv('iris.csv', names = colName)
7
8  # 撷取不同品种的鸢尾花
9  iris_setosa = iris[iris['species'] == 'Iris-setosa']
10 iris_versicolor = iris[iris['species'] == 'Iris-versicolor']
11 iris_virginica = iris[iris['species'] == 'Iris-virginica']
12 # 绘制散点图
13 plt.plot(iris_setosa['sepal_len'],iris_setosa['sepal_wd'],
14          '*',color='g',label='setosa')
15 plt.plot(iris_versicolor['sepal_len'],iris_versicolor['sepal_wd'],
16          'x',color='b',label='versicolor')
17 plt.plot(iris_virginica['sepal_len'],iris_virginica['sepal_wd'],
18          '.',color='r',label='virginica')
19 # 标注轴和标题
20 plt.xlabel('Sepal Length')
21 plt.ylabel('Sepal Width')
22 plt.title('Iris Sepal length and width anslysis')
23 plt.legend()
24 plt.show()
```

执行结果



25-7-4　鸢尾花分类统计与柱形图

如果我们想要获得不同品种鸢尾花的花瓣与花蕊的均值柱形图，首先需要统计不同品种鸢尾花的资料，这时可以使用 groupby( ) 方法。

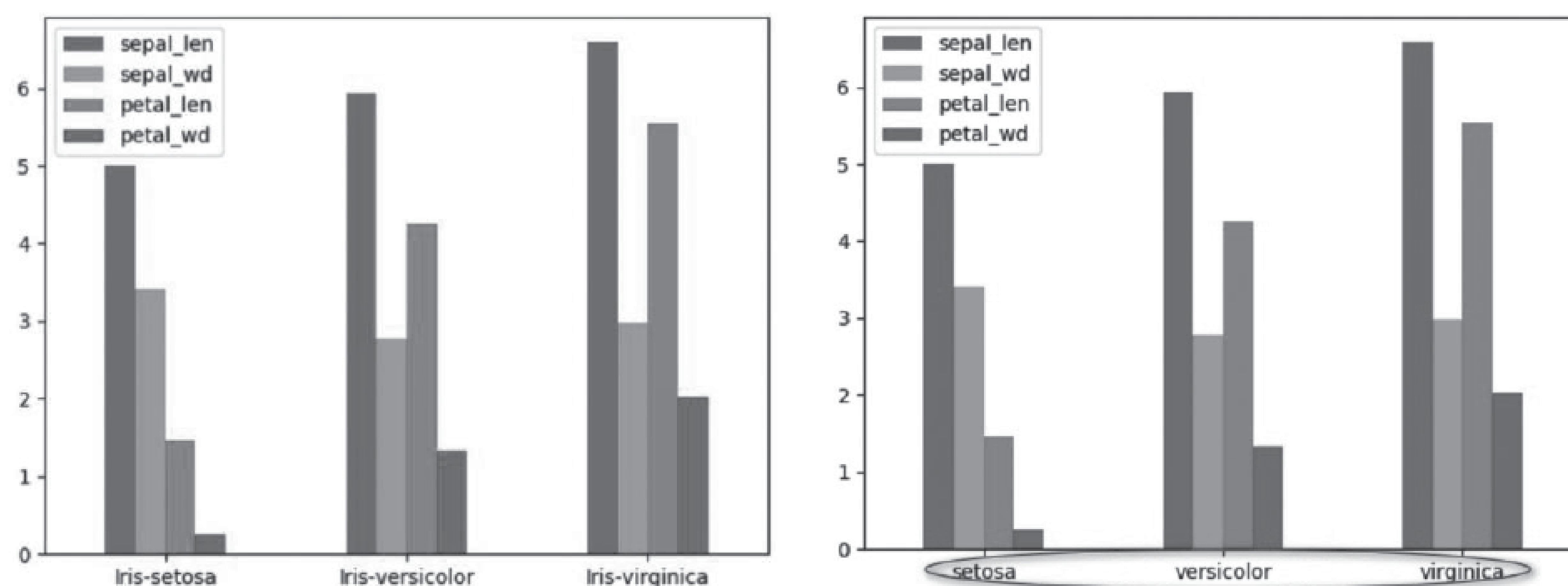
实例：延续先前实例，统计不同品种鸢尾花的花萼与花瓣的长与宽。

```
>>> iris_mean = iris.groupby('species', as_index=False).mean()
>>> print(iris_mean)
           species  sepal_len  sepal_wd  petal_len  petal_wd
0      Iris-setosa      5.006     3.418      1.464     0.244
1  Iris-versicolor      5.936     2.770      4.260     1.326
2   Iris-virginica      6.588     2.974      5.552     2.026
```

程序实例 ch25_27.py：以均值和柱形图方式绘制不同品种花萼与花瓣的长与宽。

```
1  # ch25_27.py
2  import pandas as pd
3  import matplotlib.pyplot as plt
4
5  colName = ['sepal_len','sepal_wd','petal_len','petal_wd','species']
6  iris = pd.read_csv('iris.csv', names = colName)
7
8  # 鸢尾花分组统计均值
9  iris_mean = iris.groupby('species', as_index=False).mean()
10 # 绘制柱形图
11 iris_mean.plot(kind='bar')
12 # 刻度处理
13 plt.xticks(iris_mean.index,iris_mean['species'], rotation=0)
14
15 plt.show()
```

执行结果 可以参考下方左图。



可以看到，目前品种前方字符串是 iris-，我们可以使用 apply( ) 方法将此部分字符串删除，只留下品种名称。

程序实例 ch25_28.py：重新设计上述程序，将品种前方字符串 Iris- 删除，增加下列程序代码：

```
7  iris['species'] = iris['species'].apply(lambda x: x.replace("Iris-",""))
```

执行结果

可以参考上方右图。

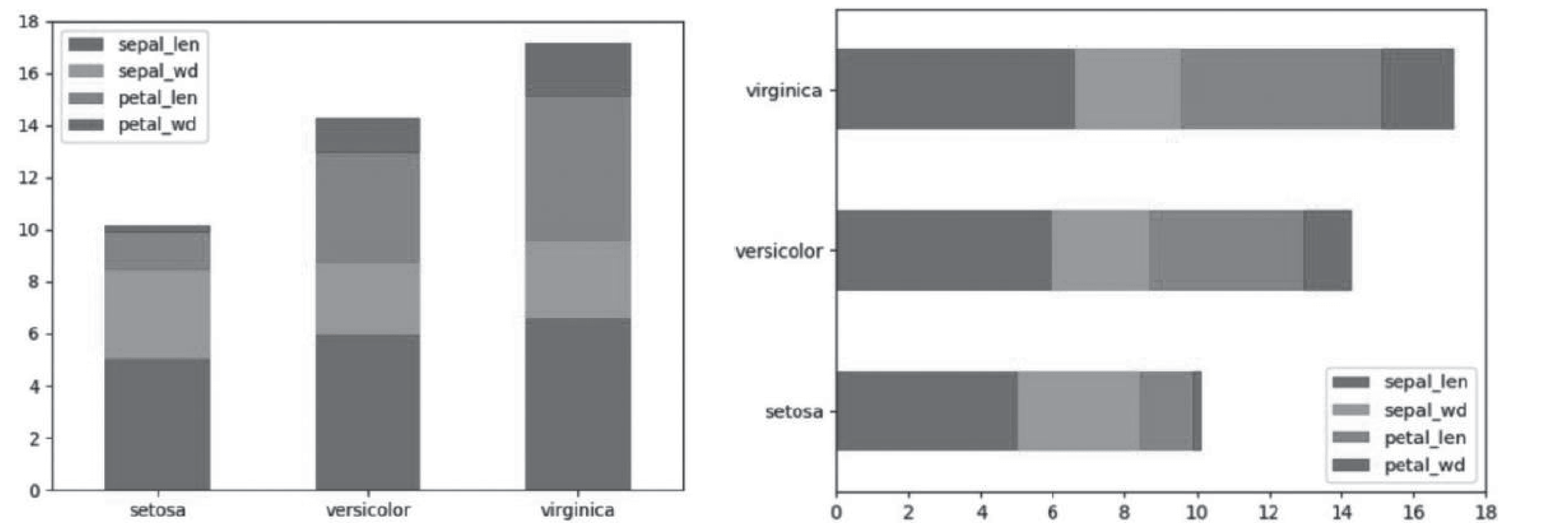
我们也可以使用堆栈方式处理上述直方图，方法是在 plot( ) 方法内增加 stacked=True。

程序实例 ch25_29.py：重新设计上述实例，但是使用堆栈方式处理数据，这个程序只有下列需修改：

```
11  iris_mean.plot(kind='bar',stacked=True)
```

执行结果

可以参考下方左图。



柱形图与条形图，差别是 bar 与 barh，可以参考下列实例。

程序实例 ch25_30.py：将前一个程序的柱形图改为条形图，这个程序只有下列需修改：

```
10  # 绘制堆栈条形图
11  iris_mean.plot(kind='barh',stacked=True)
12  # 刻度处理
13  plt.yticks(iris_mean.index,iris_mean['species'], rotation=0)
```

执行结果

可以参考上方右图。

习题

1. 以下是 2021—2025 年来台旅游统计信息，单位是万人，请做成 Series 对象，索引值必须是年份，然后打印。（25-1 节）

2021：400，2022：420，2023：450，2024：480，2025 年：500

```
===================== RESTART: D:/Python/ex/ex25_1.py =====================
2021    400
2022    420
2023    450
2024    480
2025    500
dtype: int64
```

2. 假设全球各大洲人口如下所示：(25-2 节)

North America：3.8 亿人　　South America：6.2 亿人　　Europe：7.4 亿人

Africa：12.28 亿人　　Asia：45.45 亿人

请列出下列 DataFrame 的结果。

```
===================== RESTART: D:/Python/ex/ex25_2.py =====================
               population
North America        3.80
South America        6.20
Europe               7.40
Africa              12.28
Asia                45.45
```

3. 请扩充 ex25_2.py，增加累积字段。(25-3 节)

```
===================== RESTART: D:/Python/ex/ex25_3.py =====================
               population  Cumulative
North America        3.80        3.80
South America        6.20       10.00
Europe               7.40       17.40
Africa              12.28       29.68
Asia                45.45       75.13
```

4. 请参考 20-9-4 节，将台积电实时数据最佳五档买进卖出表，改成 DataFrame 输出。注意：不同时间点获得的答案是不一样的。(25-3 节)

```
===================== RESTART: D:\Python\ex\ex25_4.py =====================
台积电最佳五档价量表
  BVolumn     Buy    Sell  SVolumn
1     454  245.50  246.00      124
2    1088  245.00  246.50      324
3    1132  244.50  247.00     1004
4     899  244.00  247.50     1046
5     452  243.50  248.00     1157
```

5. 请参考 ch25_11.py 的数据，然后建立下列 DataFrame，在 Python Shell 窗口打印，同时将此 DataFrame 结果存入 ex25_5.csv。(25-4 节)

```
===================== RESTART: D:/Python/ex/ex25_5.py =====================
         Chinese  English  Math  Natural  Society  Total  Ranking
1           14.0     12.0  13.0     10.0     13.0   62.0      3.0
2           13.0     14.0  11.0     10.0     15.0   63.0      1.0
3           15.0      9.0  12.0      8.0     15.0   59.0      5.0
4           15.0     10.0  13.0     10.0     15.0   63.0      1.0
5           12.0     11.0  14.0      9.0     14.0   60.0      4.0
Average     13.8     11.2  12.6      9.4     14.4   61.4      NaN
```

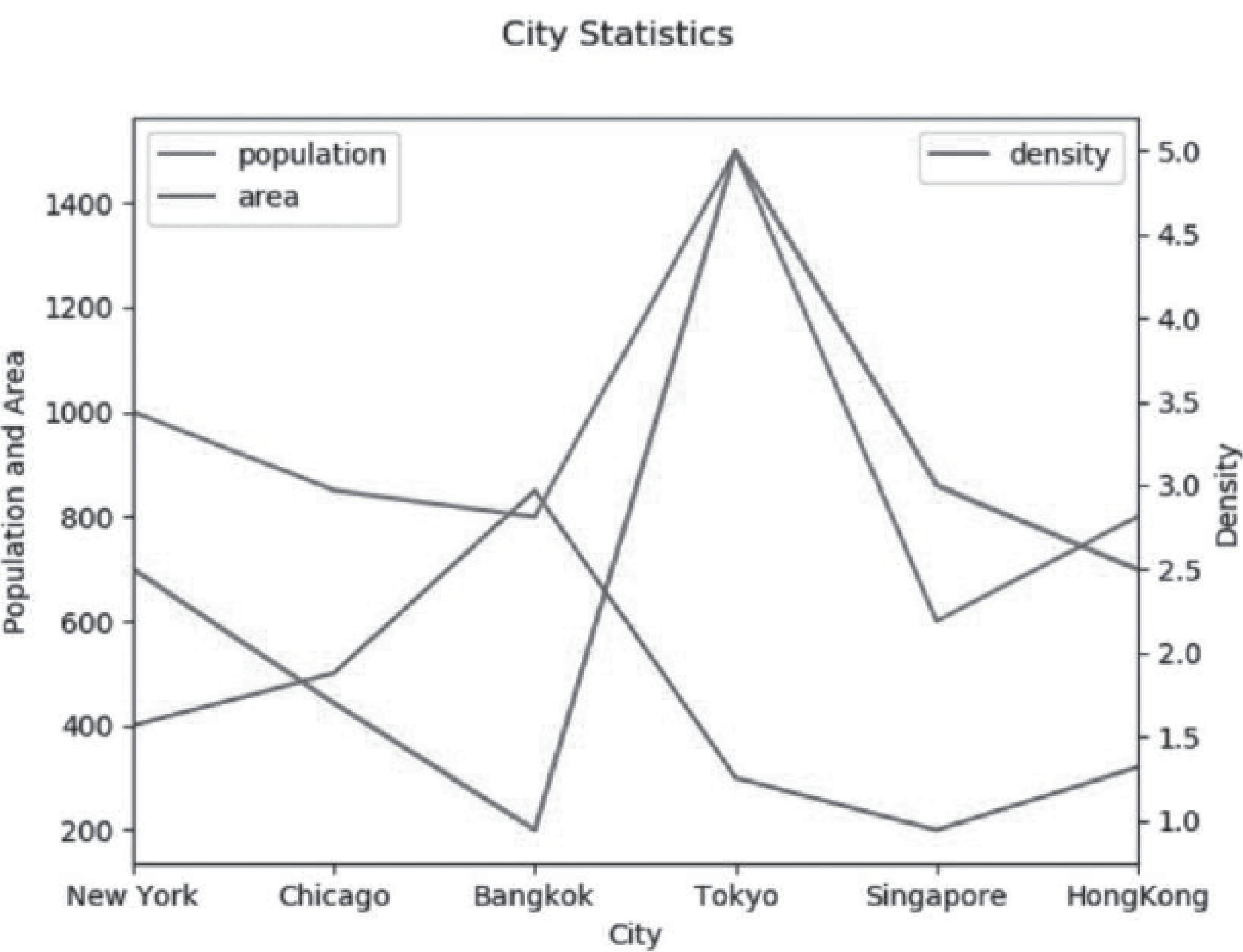
	A	B	C	D	E	F	G	H
1		Chinese	English	Math	Natural	Society	Total	Ranking
2	1	14	12	13	10	13	62	3
3	2	13	14	11	10	15	63	1
4	3	15	9	12	8	15	59	5
5	4	15	10	13	10	15	63	1
6	5	12	11	14	9	14	60	4
7	Average	13.8	11.2	12.6	9.4	14.4	61.4	

ex25_5

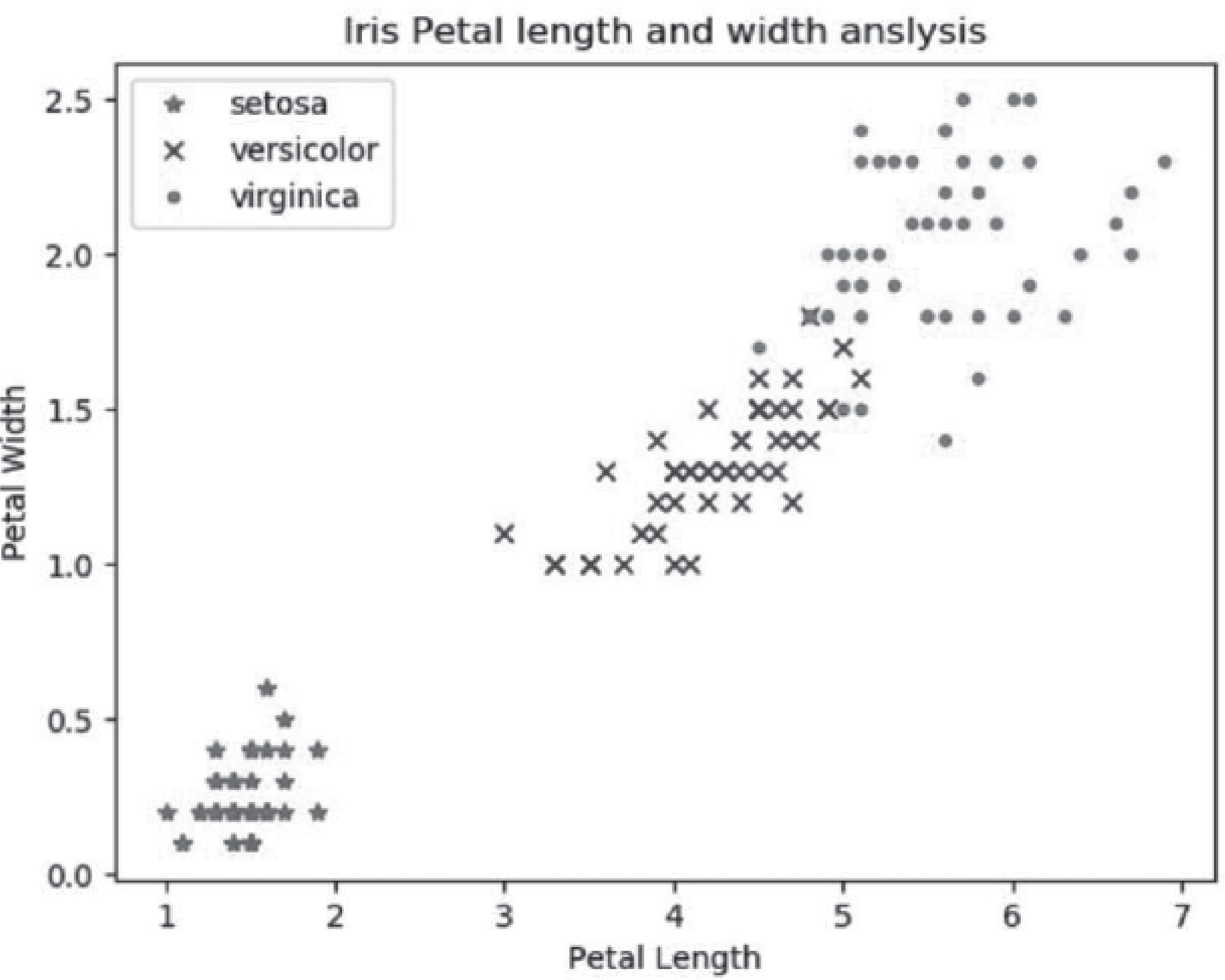
需留意 Excel 窗口（7,H）位置显示空白，这是因为 NaN 无法在 Excel 窗口显示，实际读取此 CSV 文件时，这个位置是 NaN。

6. 请参考 ch25_19.py，扩充修改方式是将 area 字段的折线图参照左边的 y 轴，增加设计 density 字段，这是 population/area 的结果，意义是每平方千米多少万人，同时 density 是使用右边自创第 2 个轴。（25-5 节）

```
===================== RESTART: D:/Python/ex/ex25_6.py =====================
           population  area   density
New York         1000   400  2.500000
Chicago           850   500  1.700000
Bangkok           800   850  0.941176
Tokyo            1500   300  5.000000
Singapore         600   200  3.000000
HongKong          800   320  2.500000
```



7. 鸢尾花专题中的 ch25_26.py 是针对花萼数据处理的散点图，请针对花瓣重新设计 ch25_26.py。（25-7 节）



A

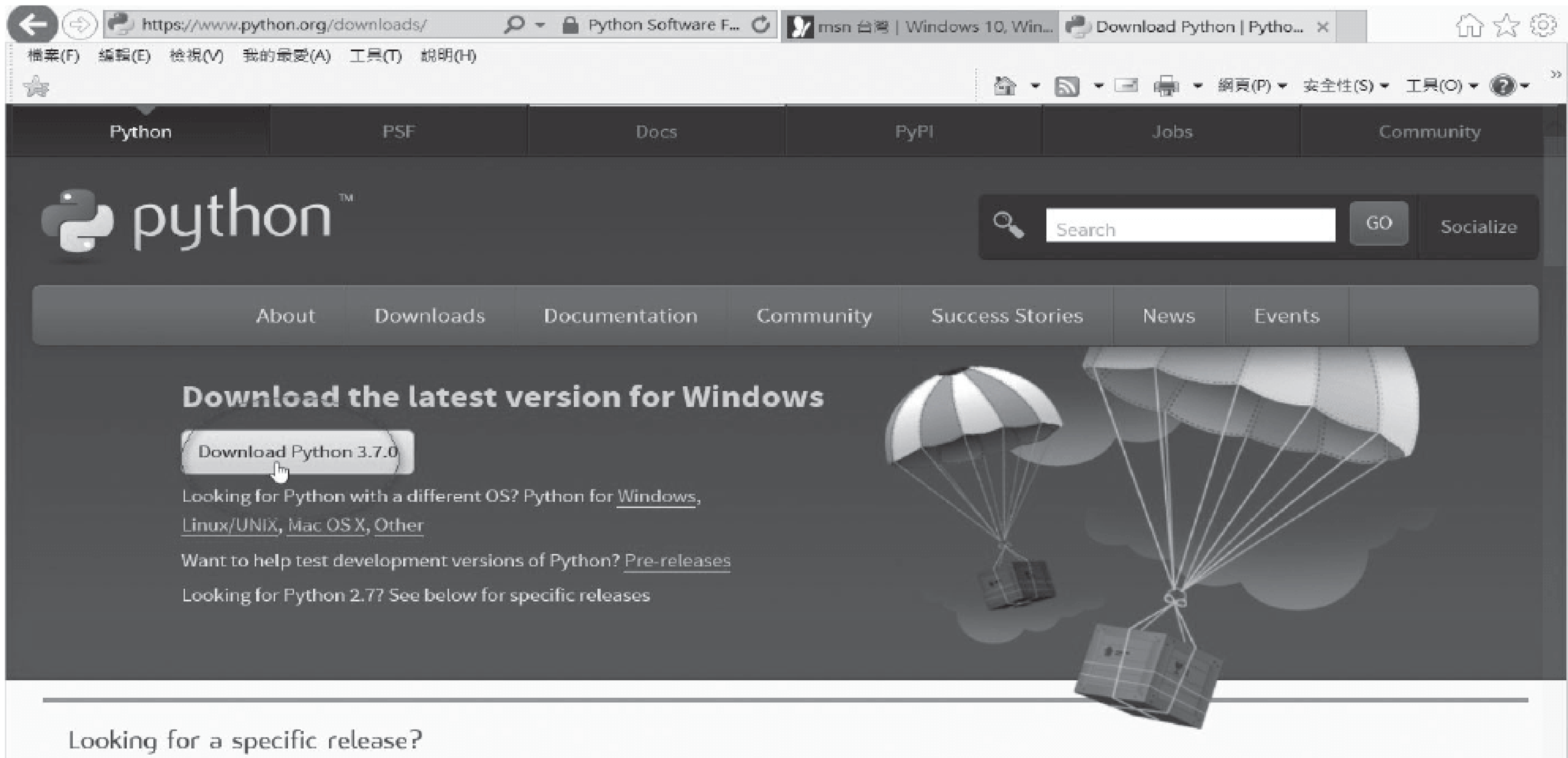
附　录　A

安装 Python

Python 安装程序在安装前会先侦测用户的计算机使用环境，然后自动协助选择安装程序。请先进入下列网页：

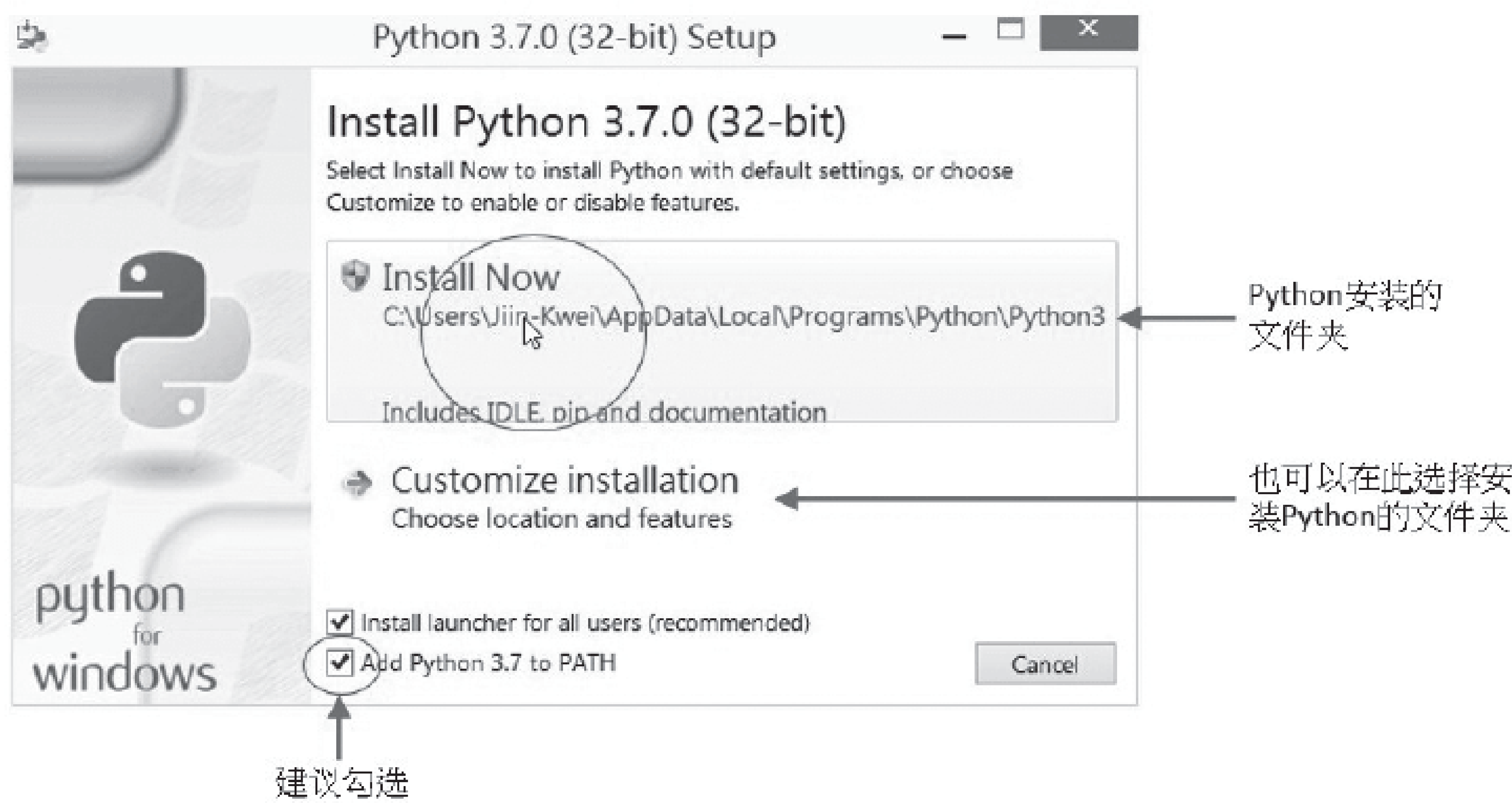
www.python.org

然后选择 Downloads 选项卡，接着可以看到 Download 按钮，笔者撰写此书时是下载的 3.7.0 版。



在 Windows 操作系统中安装 Python

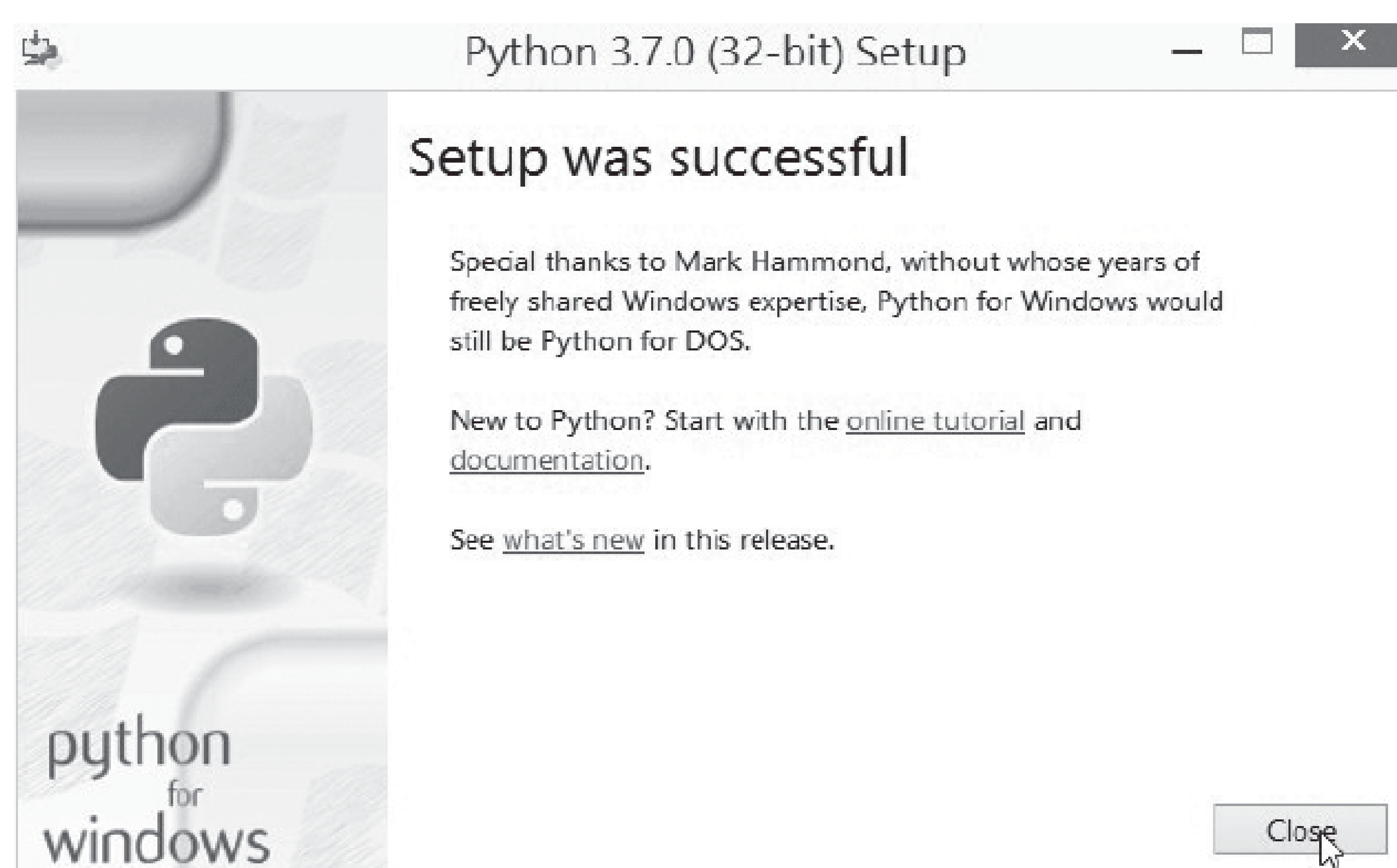
读者可以选择下载哪一个版本，此例选择下载 3.7.0 版，笔者使用 Internet Explorer 浏览器然后单击“**执行**”按钮，计算机将直接执行位于下载区的 python-3.7.exe 文件，进行安装，然后将看到下列安装画面。



注1 如果勾选 Add Python 3.7 to PATH 复选框，不论是在哪一个文件夹均可以执行 Python 可执行文件，非常方便。默认是未勾选状态，建议勾选。

注2 上述默认安装路径是在比较深层的 C:\ 文件夹路径，如果想安装在比较浅层的路径，建议选择 Customize installation，然后再选择路径，例如，选择 C:\ 即可。

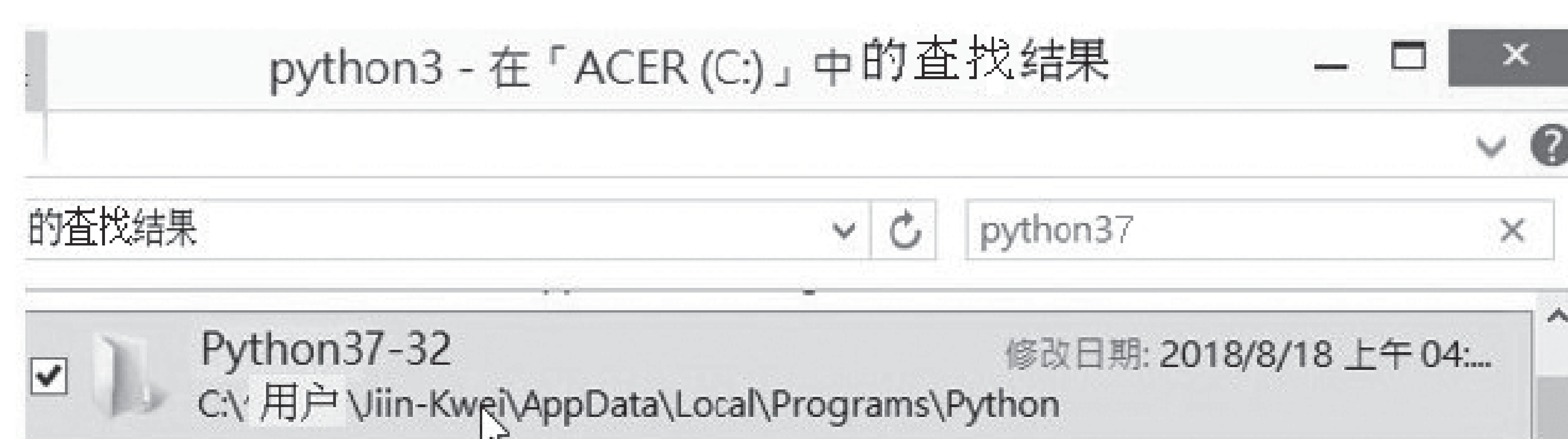
在上述界面如果单击 Install Now 选项可以直接进行安装，下方也显示了未来 Python 所在的文件夹。安装完成后将看到下列画面。



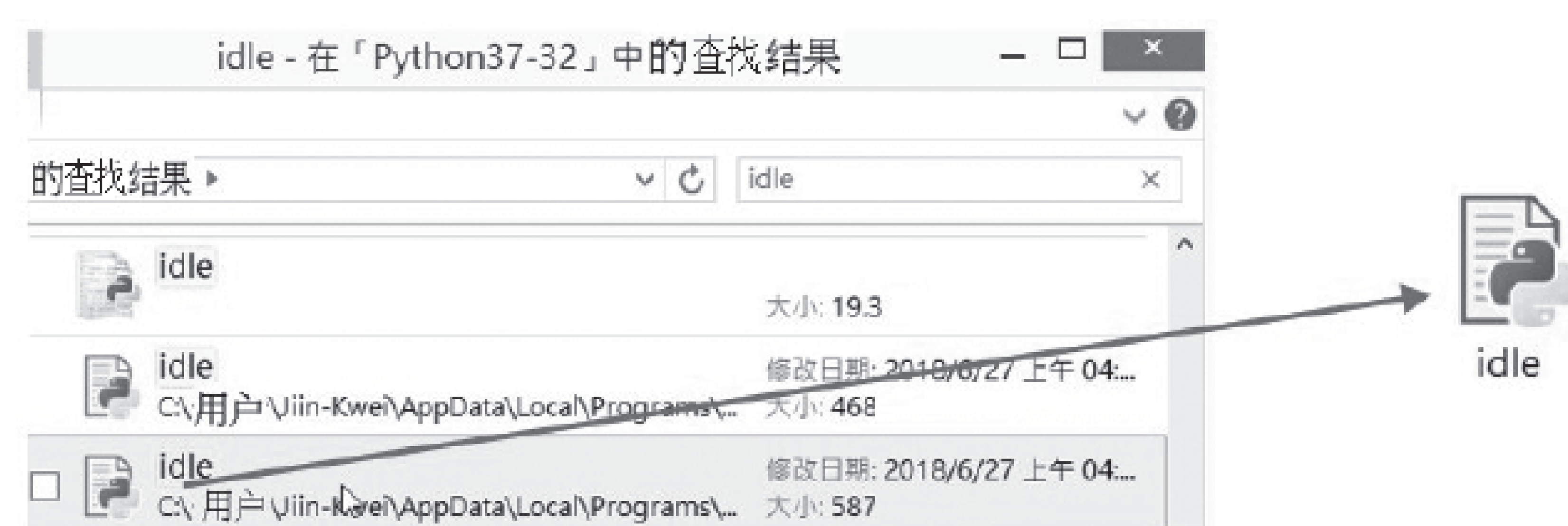
安装完成后，请进入所安装的文件夹，找到 idle 文件，这是 Python 3.7 版的整合环境程序，未来可以使用它编辑与执行 Python。

1. 查找 Python3 文件夹

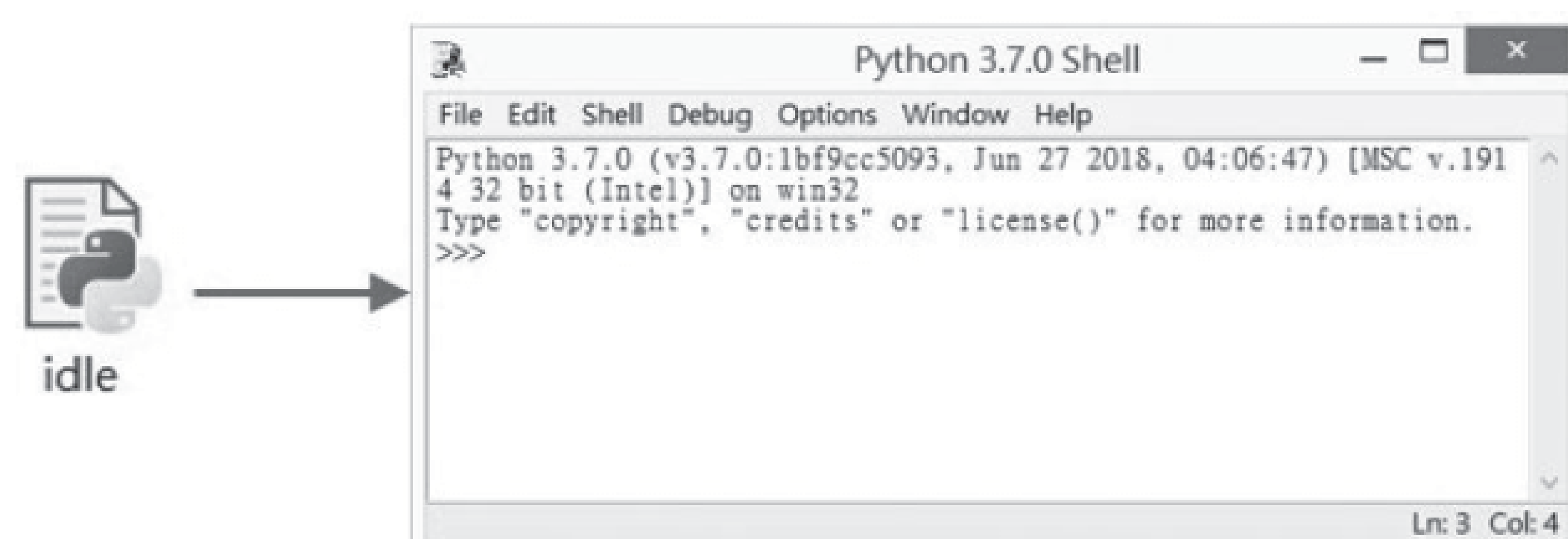
如果可以顺利进入安装 Python 的文件夹，恭喜你；如果找不到，可以打开 Windows 文件管理器，然后查找 C 文件夹，查找字符串 Python37。



Windows 操作系统会去查找与 Python3 有关的文件或文件夹，上述是找到的画面，然后单击 Python37-32（这是笔者目前的版本）。接下来是查找 Python 整合环境的 idle 程序，请在进入 Python37-32 后，在查找字段输入 idle。当找到以后，可以将此 Python 整合环境的 idle 程序拖曳至桌面。



未来只要双击 idle 图标，即可启动 Python 整合环境。



2. 查找 Python 可执行文件的路径

```
>>> import sys
>>> sys.executable
'C:\\Users\\Jiin-Kwei\\AppData\\Local\\Programs\\Python\\Python37-32\\pythonw.exe'
>>>
```

B

附　录　B

安装第三方模块

本章摘要

B-1　pip 工具
B-2　启动 DOS 与安装模块
B-3　导入模块安装更新版模块
B-4　列出所安装的模块
B-5　安装更新版模块
B-6　删除模块
B-7　查找更多模块
B-8　安装新版 pip

Python 是一个免费的软件，因此吸引了许多公司以它作为公司的官方开发语言，同时也吸引了很多公司或个人将所开发的模块放到网页上供其他人下载使用。通常我们将这些放在网络上可以下载使用的模块称为第三方模块。

B-1 pip 工具

在 Windows 操作系统中安装第三方模块需使用 pip 工具，如果是 Mac OS 或 Linux 则使用 pip3 工具。安装 Python 完成后，这些工具是放在 Scripts 目录内。

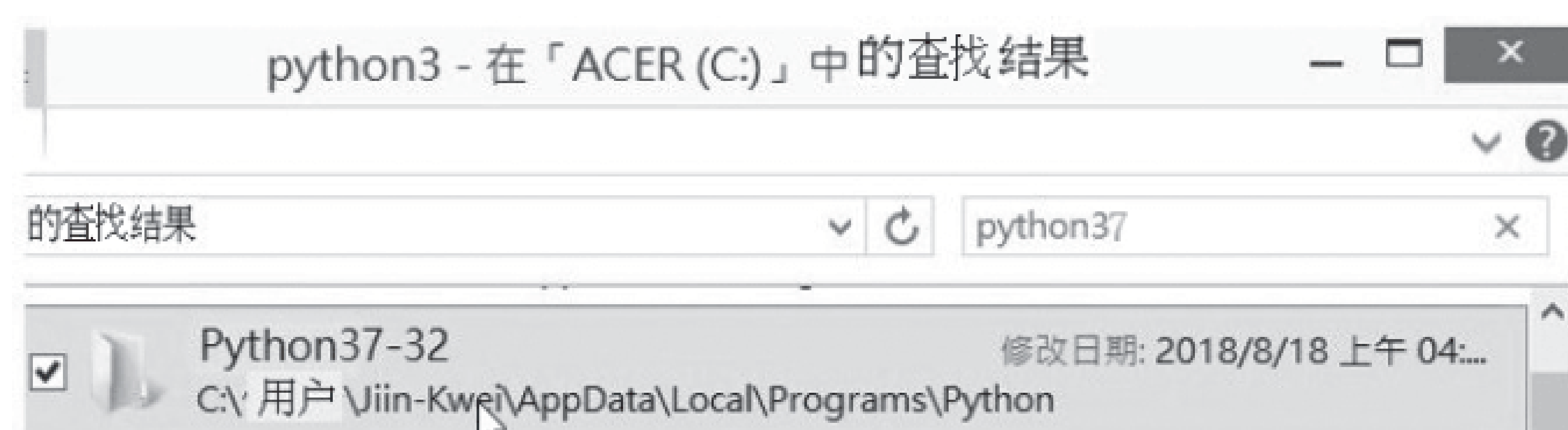
B-1-1　在 Windows 系统中将 Python 3.7 安装在 C:\

例如，如果你的 Python 3.7 版是建立在 C:\Python37-32 则非常简单，pip 工具是在下列位置。

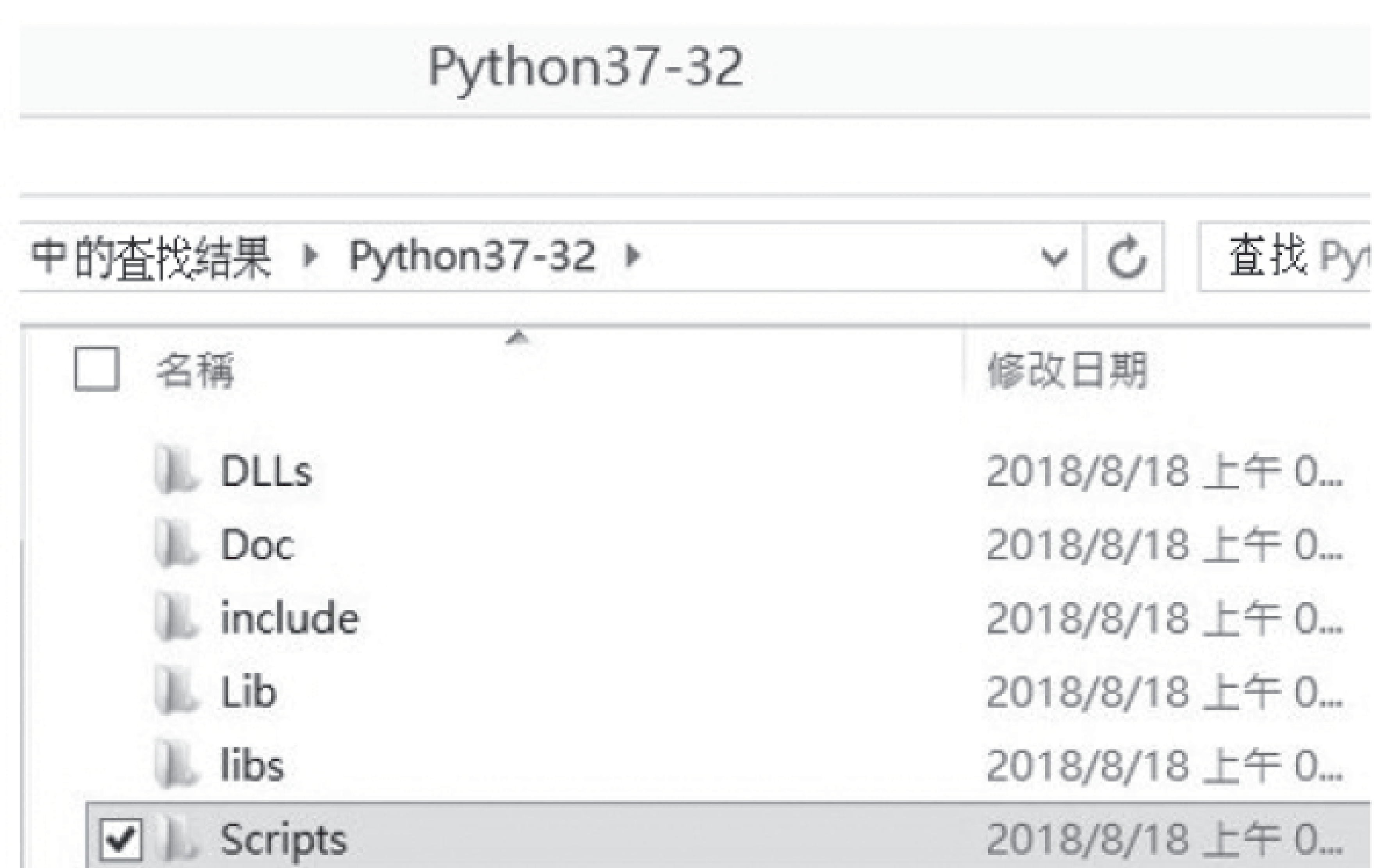
```
C:\Python37-32\Scripts\pip.exe
```

B-1-2　将 Python 3.7 安装在硬盘更深层

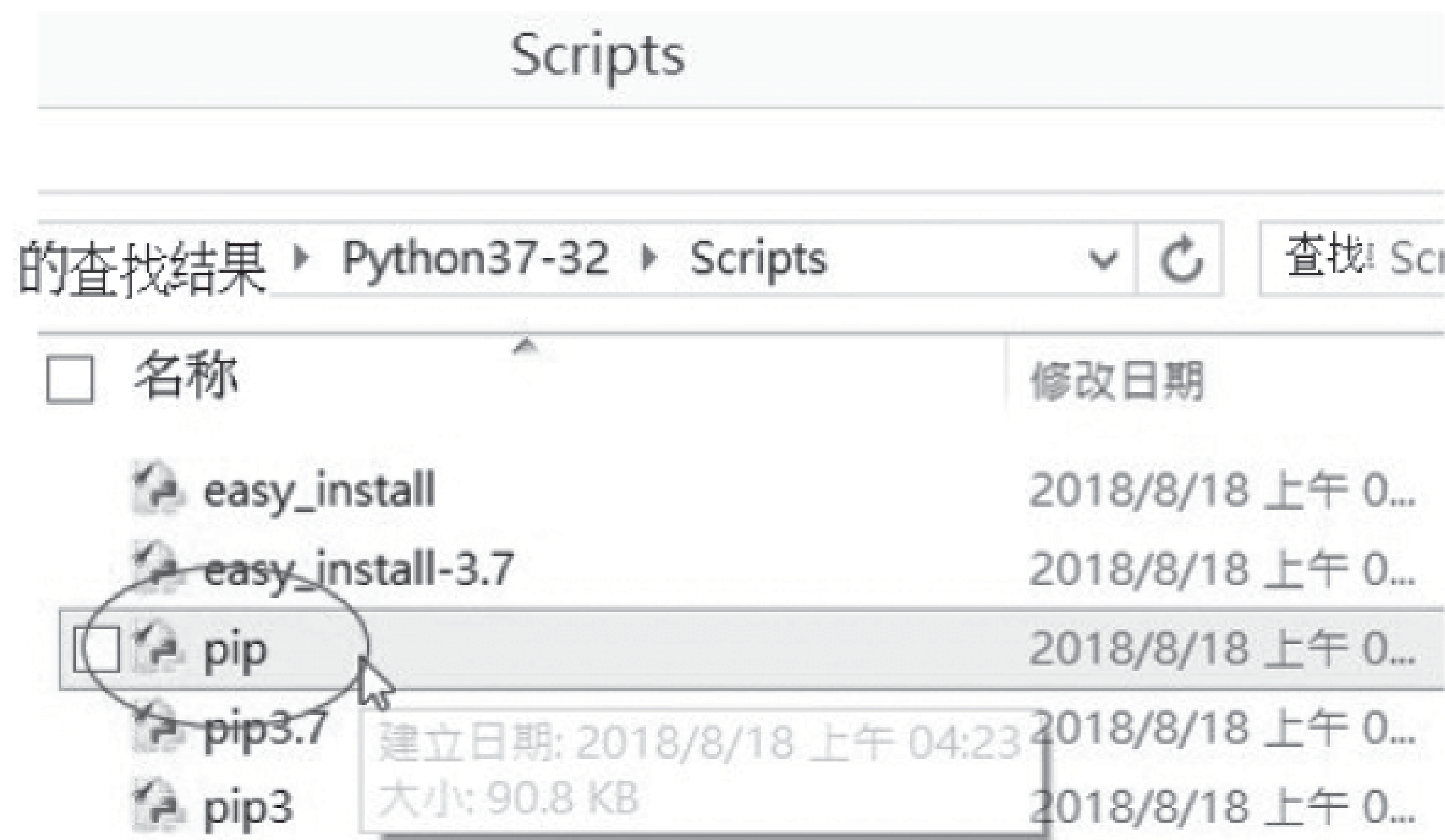
如果你的 Python 不是安装在 C:\，例如，笔者计算机是 Windows 8，安装 Python 时，在默认安装模式下 Python 是安装在下列文件夹：



这样的话整个系统会有一点儿复杂，需在查找字段输入 python37，查找此数据字符串，找到后单击 Python37-32 进入此文件夹。



单击上述 Scripts 文件夹可以进入此文件夹，然后可以看到 pip.exe 文件。



由于我们未来需进入 DOS 模式安装第三方模块，此时最好是用复制路径方式将 pip.exe 文件路径复制到 DOS 提示信息。首先单击 pip.exe 文件，单击鼠标右键执行“内容”指令，会出现 pip- 内容对话框，请选取此文件路径，如下所示。



单击鼠标右键执行“复制”指令，这时路径已经复制了，如下所示。

```
C:\Users\Jiin-Kwei\AppData\Local\Programs\Python\Python37-32\Scripts\pip.exe
```

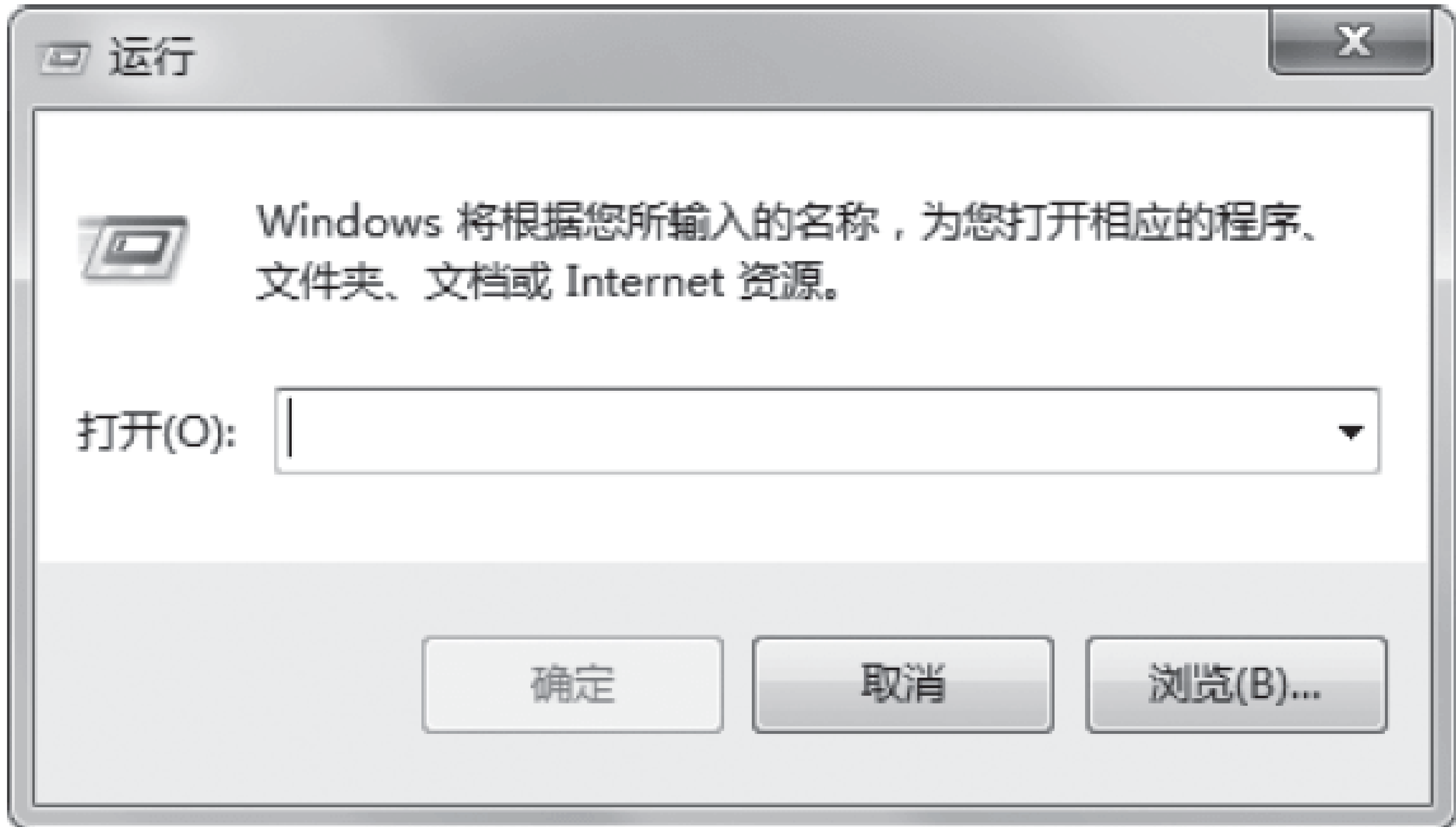
上述 Jiin-Kwei 是笔者的路径，读者应该有自己的名字路径，与笔者不同。另外，最右边的 \pip.exe 是笔者加上去的。

B-2 启动 DOS 与安装模块

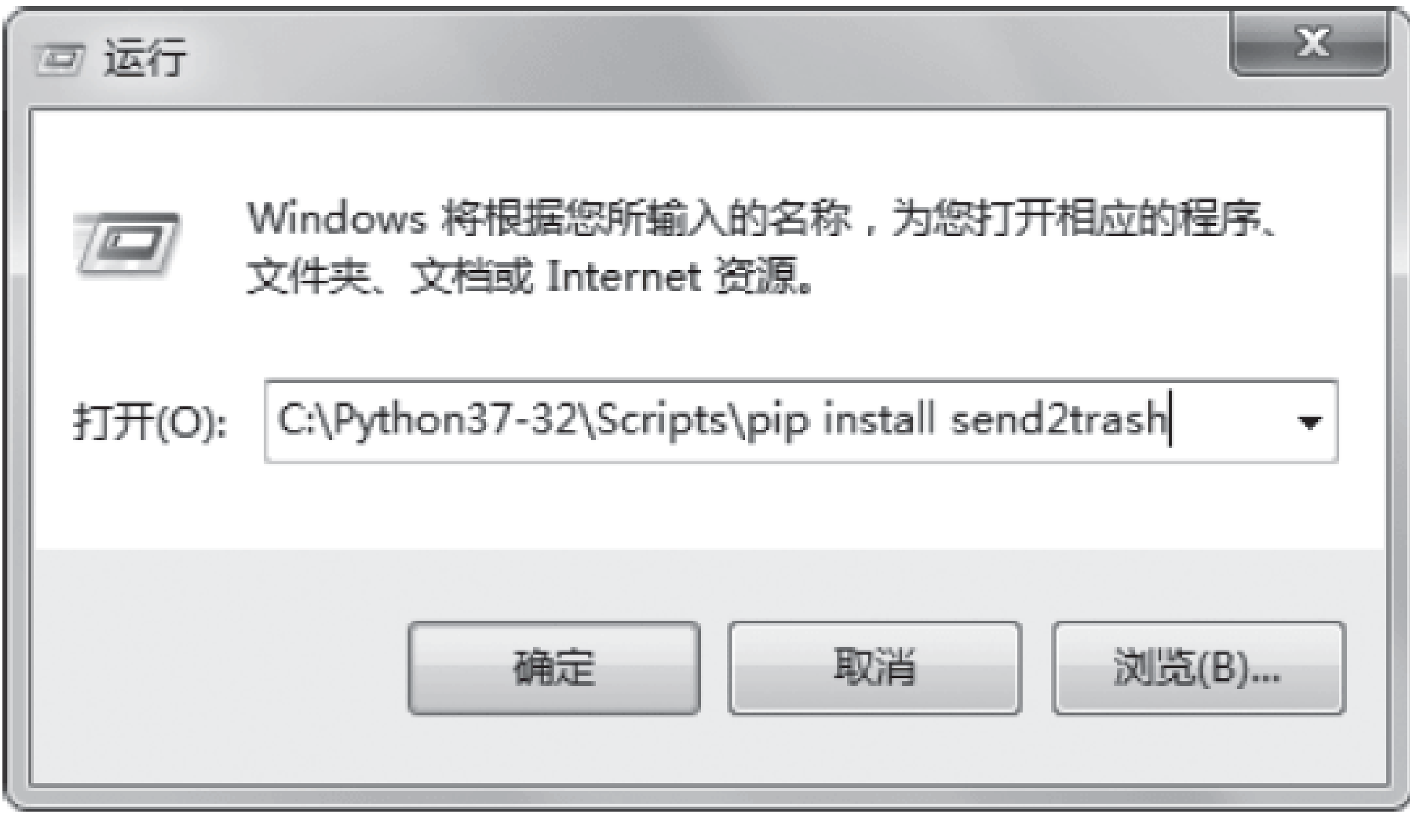
B-2-1 DOS 环境

1. 安装 Python 时没有设置 Add Python x.x to PATH

在 Windows 7 之前在“开始”菜单内可以看到“运行 ...”功能进入 DOS 环境，在 Windows 8 系统中可以按窗口键 + R 键打开 DOS 环境。接着将看到下列 DOS 的执行对话框。



这时必须将启动安装第三方模块的指令输入到“打开”字段，首先读者可以将鼠标光标移至“打开”字段，单击鼠标右键打开快捷菜单，执行“粘贴”命令，就可以将 pip.exe 的路径复制。



此时请先粘贴路径，然后在 \Scripts\ 右边输入下列指令。

```
pip install send2trash              # Windows 系统
sudo pip3 install send2trash        # Mac OS 或 Linux 系统
```

单击“确定”按钮，就可以看到 Windows 系统会另外打开 DOS 窗口执行下载安装第三方模块的画面，这个窗口会在安装完成后自动关闭。

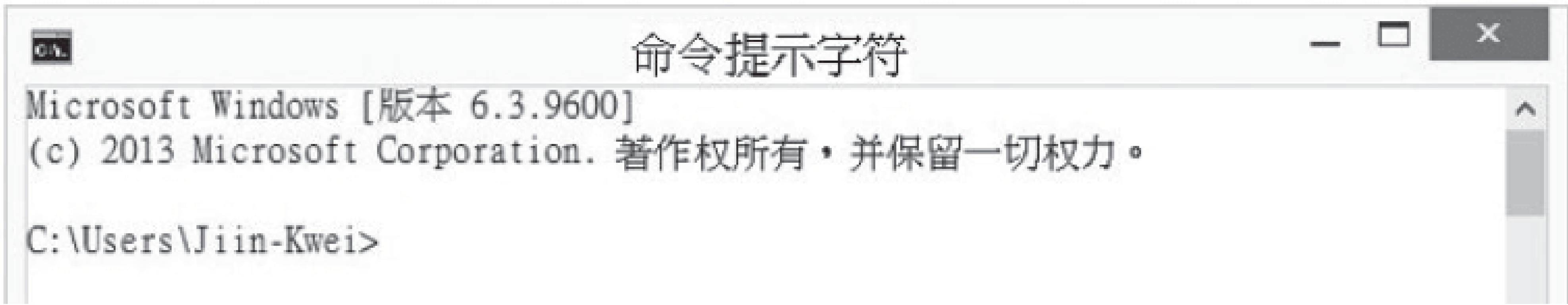
2. 安装 Python 时有设置 Add Python x.x to PATH

若是有设置 Add Python x.x（版本）to PATH，可以直接输入下列指令安装相同的模块。

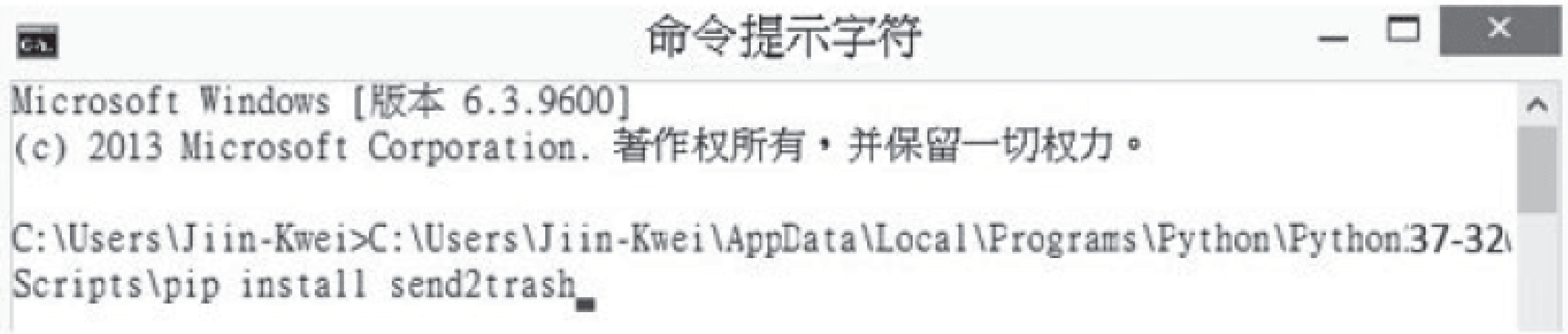
```
pip install send2trash              # Windows 系统
```

B-2-2 DOS 命令提示字符

将鼠标光标移至窗口左下角，单击鼠标右键将看到“命令提示字符”，选择“命令提示字符”可以进入此环境。



可参考 B-2-1-1 节将 pip.exe 的路径复制，再执行 pip install send2trash，即可安装第三方模块。



B-3 导入模块安装更新版模块

模块安装完成后，未来可以在程序前面执行 import 指令导入模块，同时可以测试是否安装成功，如果没有错误消息就表示安装成功了。

```
import 模块名称
import send2trash                   # 导入 send2trash 为实例
```

B-4 列出所安装的模块

可以使用 list 列出所安装的模块，如果使用‘-0’可列出有新版本的模块。

```
pip list                          # 列出所安装的模块
pip list-0                        # 列出有新版本的模块
```

B-5 安装更新版模块

未来如果有更新版，可用下列方式更新至最新版模块。

```
pip install -U 模块名称                # 更新至最新版模块
```

B-6 刪除模块

安装了模块之后，若是想删除可以使用 uninstall，例如，若是想删除 basemap，可以使用下列指令。

```
pip uninstall basemap
```

B-7 查找更多模块

可以进入 https://pypi.org。

B-8 安装新版 pip

安装好 Python 后，pip 会被自动安装，如果不小心删除可以到下列网址下载。

https://pypi.org/project/pip/

C

附 录 C

函数或方法索引表

注：索引编号是列出方法或函数所出现的章节。

D

附　　录　　D

RGB 色彩表

色彩名称	十六进制	色彩样式
AliceBlue	#F0F8FF	
AntiqueWhite	#FAEBD7	
Aqua	#00FFFF	
Aquamarine	#7FFFD4	
Azure	#F0FFFF	
Beige	#F5F5DC	
Bisque	#FFE4C4	
Black	#000000	
BlanchedAlmond	#FFEBCD	
Blue	#0000FF	
BlueViolet	#8A2BE2	
Brown	#A52A2A	
BurlyWood	#DEB887	
CadetBlue	#5F9EA0	
Chartreuse	#7FFF00	
Chocolate	#D2691E	
Coral	#FF7F50	
CornflowerBlue	#6495ED	
Cornsilk	#FFF8DC	
Crimson	#DC143C	
Cyan	#00FFFF	
DarkBlue	#00008B	
DarkCyan	#008B8B	
DarkGoldenRod	#B8860B	
DarkGray	#A9A9A9	
DarkGrey	#A9A9A9	
DarkGreen	#006400	
DarkKhaki	#BDB76B	
DarkMagenta	#8B008B	
DarkOliveGreen	#556B2F	
DarkOrange	#FF8C00	
DarkOrchid	#9932CC	

（续表）

色彩名称	十六进制	色彩样式
DarkRed	#8B0000	
DarkSalmon	#E9967A	
DarkSeaGreen	#8FBC8F	
DarkSlateBlue	#483D8B	
DarkSlateGray	#2F4F4F	
DarkSlateGrey	#2F4F4F	
DarkTurquoise	#00CED1	
DarkViolet	#9400D3	
DeepPink	#FF1493	
DeepSkyBlue	#00BFFF	
DimGray	#696969	
DimGrey	#696969	
DodgerBlue	#1E90FF	
FireBrick	#B22222	
FloralWhite	#FFFAF0	
ForestGreen	#228B22	
Fuchsia	#FF00FF	
Gainsboro	#DCDCDC	
GhostWhite	#F8F8FF	
Gold	#FFD700	
GoldenRod	#DAA520	
Gray	#808080	
Grey	#808080	
Green	#008000	
GreenYellow	#ADFF2F	
HoneyDew	#F0FFF0	
HotPink	#FF69B4	
IndianRed	#CD5C5C	
Indigo	#4B0082	
Ivory	#FFFFF0	
Khaki	#F0E68C	

（续表）

色彩名称	十六进制	色彩样式
Lavender	#E6E6FA	
LavenderBlush	#FFF0F5	
LawnGreen	#7CFC00	
LemonChiffon	#FFFACD	
LightBlue	#ADD8E6	
LightCoral	#F08080	
LightCyan	#E0FFFF	
LightGoldenRodYellow	#FAFAD2	
LightGray	#D3D3D3	
LightGrey	#D3D3D3	
LightGreen	#90EE90	
LightPink	#FFB6C1	
LightSalmon	#FFA07A	
LightSeaGreen	#20B2AA	
LightSkyBlue	#87CEFA	
LightSlateGray	#778899	
LightSlateGrey	#778899	
LightSteelBlue	#B0C4DE	
LightYellow	#FFFFE0	
Lime	#00FF00	
LimeGreen	#32CD32	
Linen	#FAF0E6	
Magenta	#FF00FF	
Maroon	#800000	
MediumAquaMarine	#66CDAA	
MediumBlue	#0000CD	
MediumOrchid	#BA55D3	
MediumPurple	#9370DB	
MediumSeaGreen	#3CB371	
MediumSlateBlue	#7B68EE	
MediumSpringGreen	#00FA9A	

（续表）

色彩名称	十六进制	色彩样式
MediumTurquoise	#48D1CC	
MediumVioletRed	#C71585	
MidnightBlue	#191970	
MintCream	#F5FFFA	
MistyRose	#FFE4E1	
Moccasin	#FFE4B5	
NavajoWhite	#FFDEAD	
Navy	#000080	
OldLace	#FDF5E6	
Olive	#808000	
OliveDrab	#6B8E23	
Orange	#FFA500	
OrangeRed	#FF4500	
Orchid	#DA70D6	
PaleGoldenRod	#EEE8AA	
PaleGreen	#98FB98	
PaleTurquoise	#AFEEEE	
PaleVioletRed	#DB7093	
PapayaWhip	#FFEFD5	
PeachPuff	#FFDAB9	
Peru	#CD853F	
Pink	#FFC0CB	
Plum	#DDA0DD	
PowderBlue	#B0E0E6	
Purple	#800080	
RebeccaPurple	#663399	
Red	#FF0000	
RosyBrown	#BC8F8F	
RoyalBlue	#4169E1	
SaddleBrown	#8B4513	
Salmon	#FA8072	

（续表）

色彩名称	十六进制	色彩样式
SandyBrown	#F4A460	
SeaGreen	#2E8B57	
SeaShell	#FFF5EE	
Sienna	#A0522D	
Silver	#C0C0C0	
SkyBlue	#87CEEB	
SlateBlue	#6A5ACD	
SlateGray	#708090	
SlateGrey	#708090	
Snow	#FFFAFA	
SpringGreen	#00FF7F	
SteelBlue	#4682B4	
Tan	#D2B48C	
Teal	#008080	
Thistle	#D8BFD8	
Tomato	#FF6347	
Turquoise	#40E0D0	
Violet	#EE82EE	
Wheat	#F5DEB3	
White	#FFFFFF	
WhiteSmoke	#F5F5F5	
Yellow	#FFFF00	
YellowGreen	#9ACD32	

E

附　录　E

ASCII 码值表

本码值表取材至 www.lookup.com 网页。

Dec	Hx	Oct	Char		Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL	(null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH	(start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX	(start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX	(end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT	(end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ	(enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK	(acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL	(bell)	39	27	047	'	'	71	47	107	G	[illegible]	103	67	147	g	g
8	8	010	BS	(backspace)	40	28	050	(	(	72	48	110	H	i	104	68	150	h	h
9	9	011	TAB	(horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF	(NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT	(vertical tab)	43	2B	053	+	+	75	4B	113	K	:	107	6B	153	k	k
12	C	014	FF	(NP form feed, new page)	44	2C	054	,		76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR	(carriage return)	45	2D	055	-		77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO	(shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI	(shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE	(data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1	(device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2	(device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3	(device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4	(device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK	(negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	Y.	(synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	E T	(end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	:A	(cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	'M	(end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	.UB	(substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC	(escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS	(file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS	(group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS	(record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US	(unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

习题及答案

第 1 章

一、是非题

1（X）：使用 Python 时需付费买授权。(1-1 节)

2（X）：Python 在执行前需要先编译，将程序转成可执行文件然后才可以执行。(1-1 节)

3（O）：Python 是面向对象的程序语言。(1-1 节)

4（X）：所有使用 Python 2 开发的软件都可以在 Python 3 上执行。(1-1 节)

5（X）：Python 在 3.0 版开始支持垃圾回收和 Unicode 功能。(1-3 节)

6（O）：可以使用 Python 设计动画游戏、动态网页设计、网络爬虫。(1-4 节)

7（X）：Python 语言的变量在使用前需要先声明。(1-4 节)

8（O）：Python 是一种动态语言，也可以称为胶水码（glue code）语言。(1-4 节)

9（O）：Python 是一种跨平台语言。(1-6 节)

二、选择题

1（D）：下列哪一个不是 Python 的特色？（1-1 节）

A. 垃圾回收　B. 直译式语言　C. 开放原始码　D. 适合简报制作

2（A）：Python 的发明与哪一个人有关？（1-2 节）

A. Guido van Rossum　B. Ross Ihaka　C. Tim Cook　D. Steve Job

3（C）：下列哪一项不是 Python 的主要应用范围？（1-4 节）

A. 设计动画游戏　B. 执行大数据分析　C. 文书编辑　D. 设计网络爬虫

4（A）：下列哪一项有关 Python 的叙述错误？（1-5 节）

A. 静态语言　B. 动态语言　C. 胶水码　D. 文字码语言

5（D）：Python 无法在下列哪一个作业环境执行？（1-6 节）

A. Windows　B. Mac OS

C. Linux　D. 以上操作系统都可以执行 Python

6（A）：下列哪一个符号不可当作 Python 的注释功能？（1-10 节）

A. @　B. #　C. '　D. "

第 2 章

一、是非题

1（O）：设计一个好的变量名称，可以方便自己与他人未来阅读程序。(2-2 节)

2（O）：为程序加上注释是程序设计的好习惯。(2-4 节)

3（O）：Python 的变量会针对所给的内容自行设置数据类型。（2-5 节）

4（X）：Python 的变量名称不可用非英文字符的其他语言。（2-6 节）

5（X）：对 Python 而言 John 与 john 算是相同的变量名称。（2-6 节）

6（O）：_5z 是 Python 合法的变量名称。（2-6 节）

7（X）：有一个函数名称是 str( )，如果使用 str 作为变量名称，将造成程序错误，然后终止执行。（2-6 节）

8（O）："%" 是用于求余数。（2-7 节）

9（X）：“//" 是用于求次方。（2-7 节）

10（X）：乘法、除法、次方的运算优先级相同，会依照出现顺序由左到右运算。（2-7 节）

11（X）："x %= y" 相当于 "x = y % x"。（2-8 节）

12（O）：下列两个公式的意义相同。（2-8 节）

```
a  /=  b
```

与

```
a  =  a  /  b
```

13（X）：有一个语句 "x, y, z = 10, 20, 30"，最后得到 x 值是 30。（2-9 节）

14（X）：有一个语句 "z = divmod（9, 5）"，可以得到 z 是 4。（2-9 节）

15（X）：del 既可当作删除变量，也可将它设为变量名称使用。（2-10 节）

16（X）：Python 允许一个语句分多行撰写，方法是在未完成语句右边加上 / 符号，Python 解释器会将下一行语句视为这一行的延伸。（2-11 节）

二、选择题

1（A）：有一程序如下：（2-2 节）

```
>>> x = 150
>>> y = 150 * 3 + 450
>>> z = 1000
>>> a = z - y
>>> a
```

上述可以得到什么输出？

A. 100　　B. 0　　C. 900　　D. 叙述语法错误

2（B）：下列哪一个是合法的变量名称？（2-6 节）

A. return　　B. _5x　　C. 9x　　D. x$d

3（C）：下列哪一个不是合法的变量名称？（2-6 节）

A. 总计　　B. _k2　　C. k,3　　D. AAA

4（C）：使用下列哪一个字符串当变量将造成程序 SyntaxError 无法执行？（2-6 节）

A. abc　　B. abs　　C. and　　D. _abc

5（D）：计算下列的 x 值。（2-7 节）

```
>>> x = 10
>>> x //= 10
>>> print(x)
```

A. 10　　B. 100　　C. 90　　D. 1

6（C）：计算下列的 x 值。（2-7 节）

```
>>> x = 11
>>> x %= 9
>>> print(x)
```

A. 10　　B. 100　　C. 2　　D. 1

7（A）：计算下列的 x 值。（2-7 节）

```
>>> x = 9 + 3 * 9 * 2 ** 2 - 30
>>> print(x)
```

A. 87　　B. 2895　　C. 46626　　D. 1

8（D）：下列指令执行结果为何？（2-9 节）

```
>>> x, y = 10, 20
>>> x, y = y, x
```

A. x=10 和 y=10　　B. x=10 和 y=10

C. x=20 和 y=20　　D. x=20 和 y=10

第 3 章

一、是非题

1（X）：如果有一个变量 x，当执行 type(x) 后得到 float，由此可以判断变量 x 是整数。（3-1 节）

2（X）：Python 语言的整数限制在 -2147483648 ～ 2147483647。（3-2 节）

3（O）：带有小数点的数字称为浮点数。（3-2 节）

4（O）：程序设计时可能发生某个变量在某一程序代码运算阶段是整数数据类型，后来在另外一个程序代码运行阶段变成字符串数据类型。（3-2 节）

5（X）：int( ) 函数可以强制将所有的字符串转成整数。（3-2 节）

6（X）：x 值是 100.5，经过 round(x) 处理，可以返回 101。（3-2 节）

7（X）：pow(x,y) 可以获得 x 开根号 y 的值。（3-2 节）

8（O）：布尔值的可能值有两种，分别是 True 和 False。（3-3 节）

9（X）：如果布尔值变量是 False，经强制 int(x) 转换，可以得到 1。（3-3 节）

10（O）：如果字符串太长想分成不同行输出，可以使用 3 个单引号包夹此字符串。（3-4 节）

11（X）：Python 允许执行字符串相加，产生新字符串。也允许字符串相减，产生新字符串。（3-4 节）

12（O）：含有 \ 的字符称为转义字符（Escape Character）。（3-4 节）

13（O）：str( ) 除了可以将数值数据转成字符串，也可以设置一个空字符串。（3-4 节）

14（X）：字符串和整数相乘将产生语法错误。（3-4 节）

15（O）：计算器内部最小的存储单位是位（bit）。（3-5 节）

16（X）：chr(x) 函数可以返回 x 的 Unicode 值。（3-5 节）

17（X）：英文大写的 ASCII 码值比英文小写的 ASCII 码值多 32。（3-5 节）

18（O）：ord(x) 函数可以返回 x 的 Unicode 值。（3-5 节）

19（O）：将 Unicode 字符串转成 bytes 数据称为编码（encode）。（3-6 节）

20（X）：utf-8 是最常使用的编码格式，这是一种固定长度的编码格式。(3-6 节)

21（O）：相同的字符串内容，应用在 bytes 数据与 Unicode 字符串使用 len( ) 函数计算长度时，可能相同也可能不相同。(3-6 节)

22（O）：将 bytes 数据转成 Unicode 字符串称为译码（decode）。(3-6 节)

23（O）："** 0.5" 具有开根号的数学效果。(3-7 节)

二、选择题

1（A）：如果有一个整数变量 x，当执行 type(x) 后可以得到什么返回值？（3-1 节）

A. int　　B. float　　C. str　　D. bool

2（B）：如果有一个浮点数变量 x，当执行 type(x) 后可以得到什么返回值？（3-1 节）

A. int　　B. float　　C. str　　D. array

3（C）：0xAA 的十进制值是多少？(3-2 节)

A. 99　　B. 100　　C. 170　　D. 200

4（D）：0b1001 的十进制值是多少？(3-2 节)

A. 3　　B. 5　　C. 7　　D. 9

5（B）：0o12 的十进制值是多少？(3-2 节)

A. 8　　B. 10　　C. 12　　D. 3

6（B）：下列哪一个函数可以将一般整数转成八进制整数？（3-2 节）

A. bin( )　　B. oct( )　　C. hex( )　　D. int( )

7（A）：下列哪一个函数可以将一般整数转成二进制整数？（3-2 节）

A. bin( )　　B. oct( )　　C. hex( )　　D. int( )

8（C）：下列哪一个函数可以将一般整数转成十六进制整数？（3-2 节）

A. bin( )　　B. oct( )　　C. hex( )　　D. int( )

9（A）：round(4.5) 的值是多少？(3-2 节)

A. 4　　B. 5　　C. True　　D. False

10（D）：有一个科学记数是 1.2E+5，它的值是多少？（3-2 节）

A. 120.0　　B. 1.2　　C. 12000.0　　D. 120000.0

11（C）：987.653 的科学记数表示是什么？（3-2 节）

A. 987.653E+2　　B. 9.87e+2　　C. 9.87653E+2　　D. 9.87653e-2

12（D）：如果有一个布尔值变量 x，当执行 type(x) 后可以得到什么返回值？（3-3 节）

A. int　　B. float　　C. str　　D. bool

13（C）：如果有一个字符串变量 x，当执行 type(x) 后可以得到什么返回值？(3-4 节)

A. int　　B. float　　C. str　　D. array

14（A）：下列哪一个转义字符（Escape Character）可以让下次输出时跳到下一行输出？(3-4 节)

A. \n　　B. \f　　C. \t　　D. \b

15（B）：哪一个转义字符（Escape Character）可以让下次输出时跳到下一页输出？（3-4 节）

A. \n　　B. \f　　C. \t　　D. \b

16（C）：在字符串前加上什么字符可以防止转义字符（Escape Character）被转译？（3-4 节）

A. a　　B. n　　C. r　　D. t

17（C）：可以在字符串与整数间用下列哪一个符号达到字符串复制效果？（3-4 节）

A. +　　B. -　　C. *　　D. /

18（C）：Unicode 码值是用什么开头？（3-5 节）

A. \m　　B. \h　　C. \u　　D. \b

19（B）：下列哪一个函数可以返回字符的 Unicode 码值？（3-5 节）

A. chr( )　　B. ord( )　　C. hex( )　　D. id( )

20（C）：bytes 数据是用什么开头？（3-6 节）

A. r'　　B. a'　　C. b'　　D. 'h

21（A）：哪个函数可以将 Uncode 字符串转成 bytes 数据？（3-6 节）

A. encode( )　　B. decode( )　　C. zip( )　　D. unzip( )

22（B）：哪个函数可以将 bytes 数据转成 Uncode 字符串？（3-6 节）

A. encode( )　　B. decode( )　　C. zip( )　　D. unzip( )

第 4 章

一、是非题

1（O）：help( ) 函数可以列出其他函数的使用说明。(4-1 节)

2（X）：%o 是格式化二进制输出。(4-2 节)

3（O）：%h 是格式化十六进制输出。(4-2 节)

4（O）：%e 与 %E 都是用于格式化科学记数的输出。(4-2 节)

5（X）：%-5d，其中负号（-）主要是格式化整数输出时，碰上负数需要输出负号（-）。(4-2 节)

6（O）：%+5d，其中正号（+）主要是格式化整数输出时，碰上正数需要输出正号（+）。(4-2 节)

7（O）：print( ) 函数内配合使用 format( ) 时，输出格式区内的变量使用 { } 表示。(4-2 节)

8（X）：print( ) 函数只能将数据输出至屏幕。(4-2 至 4-3 节)

9（X）：使用 input( ) 函数读取数字数据时，用 type( ) 函数列出所读取的数据，可以得到 int 的结果。(4-4 节)

二、选择题

1（A）：下列哪一个函数可以列出特定函数的使用说明？(4-1 节)

A. help( )　　B. print( )　　C. input( )　　D. dir( )

2（B）：print( ) 函数的哪一个参数可以设置各个数据间的分隔字符？(4-2 节)

A. value　　B. sep　　C. end　　D. file

3（C）：print( ) 函数的哪一个参数可以设置下次 print( ) 数据输出时不要换行输出？（4-2 节）

A. value　　B. sep　　C. end　　D. file

4（A）：下列哪一个可用于格式化整数输出？（4-2 节）

A. %d　　B. %f　　C. %s　　D. %h

5（B）：下列哪一个可用于格式化浮点数输出？（4-2 节）

A. %d　　B. %f　　C. %s　　D. %h

6（C）：下列哪一个可用于格式化字符串输出？（4-2 节）

A. %d　　B. %f　　C. %s　　D. %h

7（A）：使用 print( ) 配合 format( ) 时，哪一个参数可以设置靠右对齐输出？（4-2 节）

A. >　　B. <　　C. ^　　D. !

8（B）：使用 print( ) 配合 format( ) 时，哪一个参数可以设置靠左对齐输出？（4-2 节）

A. >　　B. <　　C. ^　　D. !

9（C）：使用 print( ) 配合 format( ) 时，哪一个参数可以设置居中对齐输出？（4-2 节）

A. >　　B. <　　C. ^　　D. !

10（D）：print( ) 函数的哪一个参数可以设置输出至一般文件？（4-3 节）

A. value　　B. sep　　C. end　　D. file

11（A）：使用 open( ) 打开文件时，mode 参数是下列哪一个，可以设置所打开文件只能读取？（4-3 节）

A. "r"　　B. "w"　　C. "a"　　D. "x"

12（B）：使用 open( ) 打开文件时，mode 参数是下列哪一个，可打开文件供写入，如果原先文件有内容将被覆盖？（4-3 节）

A. "r"　　B. "w"　　C. "a"　　D. "x"

13（C）：使用 open( ) 打开文件时，mode 参数是下列哪一个，可打开文件供写入，如果原先文件有内容，新写入数据将附加在后面。（4-3 节）

A. "r"　　B. "w"　　C. "a"　　D. "x"

14（D）：使用 open( ) 打开文件时，mode 参数是下列哪一个，可打开一个新的文件供写入，如果所打开的文件已经存在会产生错误？（4-3 节）

A. "r"　　B. "w"　　C. "a"　　D. "x"

15（B）：哪一个函数可以计算字符串 5*100-30，然后返回 120？（4-5 节）

A. exec( )　　B. eval( )　　C. input( )　　D. print( )

16（D）：下列哪一个函数可以列出所有 Python 所提供的内建函数？（4-6 节）

A. help( )　　B. print( )　　C. input( )　　D. dir( )

第 5 章

一、是非题

1（X）："=" 是关系运算符的等于。（5-1 节）

2（X）："&&" 是逻辑运算符的 AND。（5-2 节）

3（O）：下列变量 x 会返回 True。（5-2 节）

```
>>> x = (10 < 8) or (10 < 20)
```

4（O）：下列变量 x 会返回 False。（5-2 节）

```
>>> x = (10 > 8) and (10 > 20)
```

5（X）：Python 是使用内缩方式表达 if 语句内的程序区块，一定要内缩 4 格字符空间程序才可以运行。（5-3 节）

6（O）：Python 的 if … else 语句最大的特色是，条件判断不论是 True 或 False 均可设计一个程序代码区块供执行。（5-4 节）

7（O）：今天是星期日，假设要读者设计输入 N 天后，然后程序可以输出星期几信息，这类问题适合使用 if … elif … else 语句。（5-5 节）

8（O）：所谓的嵌套 if 语句是指 if 语句内有其他 if 语句。（5-6 节）

二、选择题

1（D）：下列哪一个是不等于关系运算符？（5-1 节）

A. >=　　B. <>　　C. <=　　D. !=

2（B）：有一个运算如下：

```
x = A op B
```

如果 A 是 True，B 是 False，结果打印 x 是 True，则 op 是什么？（5-2 节）

A. and　　B. or　　C. not　　D. ==

3（A）：哪一个语句可以用一行完成撰写？（5-3 节）

A. if 语句　　B. if … else 语句

C. if … elif … else 语句　　D. 以上皆非

4（B）：如果设计一个程序读取输入数字，如果数字大于或等于 100 输出大，如果数字小于 100 输出小，下列哪一个语句最适合设计这个程序？（5-4 节）

A. if　　B. if … else　　C. if … elif … else　　D. 嵌套 if

5（C）：如果设计一个程序读取输入 3 个苹果的重量，如果大于或等于 1.5kg 输出“A 级货”，如果小于 1.5kg 但是大于或等于 1.0kg 输出“B 级货”，其他则输出“C 级货”，下列哪一个语句最适合设计这个程序？（5-5 节）

A. if　　B. if … else　　C. if … elif … else　　D. 嵌套 if

第 6 章

一、是非题

1（X）：列表（list）是由相同数据类型的元素所组成的。（6-1 节）

2（X）：在列表（list）中元素是从索引值 1 开始配置的。（6-1 节）

3（O）：下列两个列表定义，意义相同。（6-1 节）

```
x = [1, 3, 5]
```

或

```
x = [1,3,5,]
```

4（O）：列表切片（list slices）的概念中，[:n] 可以取得列表前 n 名元素。（6-1 节）

5（X）：列表切片（list slices）的概念中，[n:] 可以取得列表后 n 名元素。（6-1 节）

6（O）：如果列表的索引是 -1，代表这是最后一个元素。（6-1 节）

7（X）：max( ) 和 min( ) 不可应用在列表元素为字符串的情况。（6-1 节）

8（O）：sum( ) 不可应用在列表元素为字符串的情况。（6-1 节）

9（O）：有两个列表 x 和 y，可以执行 x + y。（6-1 节）

10（X）：有两个列表 x 和 y，可以执行 x * y。（6-1 节）

11（O）：有一个 Python 程序内容如下：(6-1 节)

```
x = ['big', 'small', 'medium']
del x[0]
print(x)
```

可以得到下列结果。

```
['small', 'medium']
```

12（O）：del 可以用于删除列表元素，也可以删除整个列表。(6-1 节)

13（x）：有一个 Python 指令片段如下：(6-2 节)

```
>>> x = "i love python"
>>> y = x.title( )
```

若是打印 y，可以得到 "I love python"。

14（O）：有一个 Python 程序如下：(6-2 节)

```
x = ['big', 'small', 'medium']
y = x[0].lower( )
print(y)
```

可以得到下列结果。

```
big
```

15（O）：strip( ) 可以删除字符串头尾两边多余的空白。(6-2 节)

16（X）：append( ) 可以在列表开头增加元素。(6-4 节)

17（X）：insert( ) 主要是在列表末端插入元素。(6-4 节)

18（O）：pop( ) 除了可以删除元素，也可以将所删除的元素返回。(6-4 节)

19（X）：remove( ) 可以删除指定索引位置的元素。(6-4 节)

20（X）：sort( ) 排序是由大排到小。(6-5 节)

21（X）：sorted( ) 排序可以造成列表元素顺序永久更改。(6-5 节)

22（O）：有一个 Python 程序如下：(6-6 节)

```
str = ['2', '2', '3', '3', '2']
search_str = '2'
i = str.index(search_str)
print(i)
```

可以得到 i 是 0。

23（X）：有一个 Python 程序如下：(6-7 节)

```
num = [[1,2,3,4],[5,6,7,8]]
i = num[1][1]
print(i)
```

可以得到 i 是 2。

24（O）：len( ) 方法除了可以计算列表（list）元素个数，也可以用于计算字符串（string）长度。(6-9 节)

25（X）：将字符串转成列表时，原先字符串的空格符部分将被舍去。(6-9 节)

26（X）：list( ) 可以将字符串转成列表，split( ) 也可以将字符串转成列表，它们的用法是相同的，结果也是相同的。(6-9 节)

27（X）：Python 的 in 表达式主要是比较两个对象是否相同。(6-10 节)

28（O）：在 Python 语言中，两个不同地址的对象，即使内容相同，使用 is 指令时，会被视为不同的对象。(6-11 节)

二、选择题

1（A）：使用列表（list）时，如果索引值是下列哪一个，代表这是列表的最后一个元素？（6-1 节）

A. −1　　B. 0　　C. 1　　D. max

2（B）：有一个 Python 程序如下：(6-1 节)

```
x = ['big', 'small']
x = x * 3
print(x)
```

可以得到下列哪一个结果？

A. 程序错误

B. ['big', 'small', 'big', 'small', 'big', 'small']

C. ['big', 'small']

D. ['big', 'big', 'big', 'small', 'small', 'small']

3（D）：有一个 Python 程序如下：(6-1 节)

```
x = ['big', 'small', 'medium']
x[1] = 'size'
print(x)
```

可以得到下列哪一个结果？

A. ['big', 'small', 'medium']　　B. ['big', 'small', 'size']

C. ['size', 'small', 'medium']　　D. ['big', 'size', 'medium']

4（D）：有一个 Python 程序如下：(6-1 节)

```
x = ['big', 'small', 'medium']
y = len(x)
print(y)
```

可以得到下列哪一个结果？

A. 0　　B. 1　　C. 2　　D. 3

5（B）：有一个 Python 程序如下：(6-2 节)

```
x = ['big', 'small', 'medium']
y = x[2].title()
print(y)
```

可以得到下列哪一个结果？

A. BIG　　B. Medium　　C. Small　　D. MEDIUM

6（C）：有一个 Python 程序如下：(6-2 节)

```
x = ' Silicon Stone '
y = x.rstrip()
print("/%s/" % y)
```

可以得到下列哪一个结果？

A. / Silicon Stone /　　B. /Silicon Stone/

C. / Silicon Stone/　　D. /Silicon Stone /

7（A）：下列哪一个指令可以列出对象的所有方法？（6-2 节）

A. dir　　B. help　　C. display　　D. list

8（C）：有一程序如下：(6-2 节)

```
>>> n = 4
>>> y = n.bit_length()
```

上述 y 的值是多少？

A. 1　　B. 2　　C. 3　　D. 4

9（C）：有一个 Python 程序如下：(6-4 节)

```
x = [ ]
x.append('big')
x.append('small')
print(x)
x.append('medium')
```

可以得到下列哪一个结果？

A. ['big', 'small', 'medium']　　B. ['big']

C. ['big', 'small']　　D. []

10（D）：有一个 Python 程序如下：(6-4 节)

```
x = ['big', 'small', 'medium', 'large']
y = x.pop( )
print(y)
```

可以得到下列哪一个结果？

A. big　　B. small　　C. medium　　D. large

11（A）：有一个 Python 程序如下：(6-4 节)

```
x = ['big', 'small', 'medium', 'large']
y = 'big'
x.remove(y)
print(x)
```

可以得到下列哪一个结果？

A. ['small', 'medium', 'large']　　B. ['big', 'medium', 'large']

C. ['big', 'size', 'medium']　　D. ['big', 'small', 'medium']

12（A）：有一个 Python 程序如下：(6-5 节)

```
x = ['big', 'small', 'medium', 'large']
x.reverse()
print(x)
```

可以得到下列哪一个结果？

A. ['large', 'medium', 'small', 'big']

B. ['big', 'small', 'medium', 'large']

C. ['big', 'large', 'medium', 'small']

D. ['small', 'medium', 'large', 'big']

13（B）：[::-1] 的意义。(6-5 节)

A. sort( )　　B. reverse( )　　C. sort(reverse=True)　　D. sorted( )

14（C）：有一个 Python 程序如下：(6-5 节)

```
x = ['big', 'small', 'medium', 'large']
x.sort( )
print(x)
```

可以得到下列哪一个结果？

A. ['large', 'medium', 'small', 'big']

B. ['big', 'small', 'medium', 'large']

C. ['big', 'large', 'medium', 'small']

D. ['small', 'medium', 'large', 'big']

15（D）：有一个 Python 程序如下：(6-5 节)

```
x = ['big', 'small', 'medium', 'large']
x.sort(reverse=True)
print(x)
```

可以得到下列哪一个结果？

A. ['large', 'medium', 'small', 'big']

B. ['big', 'small', 'medium', 'large']

C. ['big', 'large', 'medium', 'small']

D. ['small', 'medium', 'large', 'big']

16（B）：有一个 Python 程序如下：(6-5 节)

```
x = ['big', 'small', 'medium', 'large']
y = sorted(x)
print(x)
```

可以得到下列哪一个结果？

A. ['large', 'medium', 'small', 'big']

B. ['big', 'small', 'medium', 'large']

C. ['big', 'large', 'medium', 'small']

D. ['small', 'medium', 'large', 'big']

17（B）：有一个 Python 程序如下：(6-6 节)

```
x = [1,2,3,4]
y = x.index(3)
```

y 的内容为何？

A. 1　　B. 2　　C. 3　　D. 4

18（B）：有一个 Python 程序如下：(6-7 节)

```
x = [[1,2,3],[4,5,6],[7,8,9]]
y = x[1][1]
```

y 的内容为何？

A. 2　　B. 5　　C. 8　　D. 1

19（C）：有一个 Python 程序如下：(6-7 节)

```
str1 = ['small', 'medium', 'large']
str2 = ['fast', 'slow']
str1.append(str2)
print(str1)
```

可以得到下列哪一个结果？

A. [['small', 'medium', 'large'], 'fast', 'slow']

B. ['small', 'medium', 'large', 'fast', 'slow']

C. ['small', 'medium', 'large', ['fast', 'slow']]

D. ['fast', 'large', 'medium', 'slow', 'small']

20（B）：有一个 Python 程序如下：(6-7 节)

```
str1 = ['small', 'medium', 'large']
str2 = ['fast', 'slow']
str1.extend(str2)
print(str1)
```

可以得到下列哪一个结果？

A. [['small', 'medium', 'large'], 'fast', 'slow']

B. ['small', 'medium', 'large', 'fast', 'slow']

C. ['small', 'medium', 'large', ['fast', 'slow']]

D. ['fast', 'large', 'medium', 'slow', 'small']

21（C）：有一个 Python 程序片段如下：(6-8 节)

```
>>> x = [1,2,3]
>>> y = x
>>> x.append(4)
>>> y.append(5)
```

上述 x 的内容为何？

A. [1,2,3]　　B. [1,2,3,4]　　C. [1,2,3,4,5]　　D. []

22（B）：有一个 Python 程序片段如下：(6-8 节)

```
x = [1,2,3]
y = x[:]
x.append(4)
y.append(5)
```

上述 x 的内容为何？

A. [1,2,3]　　B. [1,2,3,4]　　C. [1,2,3,4,5]　　D. []

23（C）：有一个 Python 指令片段如下：(6-9 节)

```
x = '123456789'
y = x[1:9:3]
```

上述 y 的内容为何？

A. '123'　　B. '159'　　C. '258'　　D. '369'

24（D）：有一个 Python 指令片段如下：(6-9 节)

```
>>> x = ['1','2','3']
>>> y = '4'
>>> z = y.join(x)
```

上述 z 的内容为何。

A. '1234'　　B. '123'　　C. '4123'　　D. '14243'

第 7 章

一、是非题

1（ O ）：列表（list）是一种可迭代对象（iterable object）。(7-1 节)

2（ O ）：下述语句可以产生含 'C'，'D'，'E'，'F'4 个元素的列表。(7-1 节)

```
alphabets = ['A', 'B', 'C', 'D', 'E', 'F']
for alphabet in alphabets[2:]:
    print(alphabet)
```

3（ X ）：delall( ) 可以删除列表内所有元素。(7-1 节)

4（ X ）：range( ) 函数所产生的可迭代对象称为列表（list）。(7-2 节)

5（ O ）：下述语句可以列出 1 ～ 9 的元素。(7-2 节)

```
for i in range(1, 10):print(i)
```

6（ X ）：下述语句可以列出 10 ～ 2 的元素。(7-2 节)

```
for i in range(10,1):print(i)
```

7（ X ）：当 range( ) 函数有 3 个参数时，第 2 个参数值是间隔值。(7-2 节)

8（ X ）：当 range( ) 函数有 3 个参数时，第 3 个参数值是终止值。(7-2 节)

9（ X ）：下列程序可以列出 9×9 乘法表。(7-3 节)

```
for i in range(1, 9):
    for j in range(1, 9):
        result = i * j
        print("%d*%d=%-3d" % (i, j, result), end=" ")
    print()
```

10（ O ）：break 指令可以让 for 或 while 循环中断。(7-3 和 7-4 节)

11（ X ）：凡是使用 for 语句的循环，只要直接将 for 改为 while，都可正常执行，而获得相同的结果。(7-3 和 7-4 节)

12（ X ）：有一个列表如下：

```
numlist = [1, 2, 3, 4, 5]
```

如果想要分行列出此列表的所有元素，最佳方式是使用 while 循环。(7-4 节)

13（ X ）：下列程序可以列出 1 ～ 9 的元素。(7-4 节)

```
while i in range(1, 10):
    print(i)
```

14（ O ）：下列是无限循环。(7-4 节)

```
while 1:
    pass
```

15（ O ）：enumerate 对象的每个元素都是索引与数据值所组成的。(7-5 节)

二、选择题

1（ A ）：下列哪一项目不是可迭代对象？（ 7-1 节 ）

A. 整数　　B. 列表（list）　　C. 元组（tuple）　　D. range

2（ D ）：请列出下列程序的执行结果。(7-1 节)

```
alphabets = ['A', 'B', 'C', 'D', 'E', 'F']
for alphabet in alphabets[-2:]:
    print(alphabet)
```

A. A B　　B. B C　　C. C D　　D. E F

3（A）：请列出下列程序的执行结果。(7-1 节)

```
alphabets = ['A', 'B', 'C', 'D', 'E', 'F']
for alphabet in alphabets[:2]:
    print(alphabet)
```

A. A B　　B. B C　　C. C D　　D. E F

4（D）：有一个程序片段如下，请列出执行结果。(7-2 节)

```
for x in range(5,1):
    print(x)
```

A. 5 4 3 2　　B. 1 2 3 4　　C. 4 3 2　　D. 空 range 对象

5（A）：有一个程序片段如下：(7-2 节)

```
colors = ["Red", "Green", "Blue"]
shapes = ["Circle", "Square"]
result = [[color, shape] for color in colors for shape in shapes]
for color, shape in result:
    print(color, shape)
```

第一个输出为何？

A. Red Circle　　B. Red Square　　C. Blue Square　　D. Green Circle

6（C）：有一个程序片段如下：(7-2 节)

```
colors = ["Red", "Green", "Blue"]
shapes = ["Circle", "Square"]
result = [[color, shape] for color in colors for shape in shapes]
for color, shape in result:
    print(color, shape)
```

最后一个输出为何？

A. Red Circle　　B. Red Square　　C. Blue Square　　D. Green Circle

7（B）：下列程序执行结果 total 值是多少？(7-2 节)

```
total = 0
for digit in range(1, 11):
    if digit == 5:
        break
    total += digit
print(total)
```

A. 0　　B. 10　　C. 50　　D. 55

8（C）：下列程序执行结果 total 值是多少？(7-2 节)

```
total = 0
for digit in range(1, 11):
    if digit == 5:
        continue
    total += digit
print(total)
```

A. 0　　B. 10　　C. 50　　D. 55

9（A）：下列程序执行结果 n 值是多少。（7-3 节）

```
players = [['James', 202],
           ['Curry', 193],
           ['Durant', 205],
           ['Jordan', 199],
           ['David', 211],
           ['Norton', 220],
           ['Manning', 198]]
n = 0
for player in players:
    if (player[1] <= 210) and (player[1] >= 195):
        n += 1
        continue
print(n)
```

A. 4　　B. 5　　C. 6　　D. 7

10（A）：下列程序执行结果 total 值是多少？（7-4 节）

```
n = 0
total = 0
while n <= 10:
    n += 1
    if ( n % 2 != 0 ):
        break
    total += n
print(total)
```

A. 0　　B. 1　　C. 45　　D. 55

11（D）：下列是一个无限循环，如果要中断此无限循环，可以使用下列哪一个按键？（7-4 节）

```
while True:
    print("True")
```

A. Ctrl + A　　B. Esc　　C. Enter　　D. Ctrl + C

12（A）：下列程序执行结果为何？（7-4 节）

```
index = 0
while index <= 8:
    index += 1
    if ( index % 2 == 0 ):
        continue
    print(index,end=',')
```

A. 1,3,5,7,9,　　B. 1,3,5,7,　　C. 2,4,6,8,　　D. 2,4,6,8,10,

第 8 章

一、是非题

1（X）：元组的元素值不可以更改，但是元素数量可以更改。（8-1 节）

2（O）：元组的定义是将元素放在小括号内“( )”。(8-1 节)

3（X）：设置元组的元素时，如果有多个元素，这些元素彼此用“;”隔开。(8-1 节)

4（O）：读取元组 tuple1 的第一个元素，可以使用下列语法。(8-2 节)

```
value = tuple1[0]
```

上述语句会将元组 tuple1 的第一个元素读入 value。

5（O）：当定义一个元组 x 后，未来可以重新定义此元组 x 的内容，所以下列语法不会有错误。(8-5 节)

```
>>> x = (1,2,3)
>>> x = (4,5,6)
```

6（O）：元组数据可以转成列表，列表数据也可以转成元组。(8-8 节)

7（O）：将 enumerate 对象转成列表时，此列表的元素是元组。(8-10 节)

8（X）：使用 zip( ) 打包对象时，被打包的对象长度必须相同。(8-11 节)

9（X）：使用 zip( ) 打包对象时，被打包的对象数据结构必须相同。(8-11 节)

10（O）：使用元组存储数据，可以提供更安全的保护，避免因疏忽造成数据被更改。(8-14 节)

11（X）：存取元组的元素比存取列表元素要更花时间。(8-14 节)

二、选择题

1（D）：下列哪一个数据类型不可以当作元组的元素？（8-1 节）

A. 整数　　B. 字符

C. 列表　　D. 以上皆可当作元组元素

2（A）：定义元组时是使用小括号 ( )，读取元组索引值时是使用哪一项？（8-2 节）

A. ( )　　B. [ ]　　C. { }　　D. 以上皆可

3（B）：下列哪一项叙述正确？（8-3 ~ 8-7 节）

A. for 循环不可以应用在元组　　B. 元组内容不可修改

C. append( ) 可以应用在元组　　D. pop( ) 可以应用在元组

4（B）：有一个片段指令如下，请列出执行结果。(8-6 节)

```
>>> fruits = ('apple', 'orange', 'banana', 'watermelon', 'grape')
>>> print(fruits[-2:])
```

A. ('apple', 'orange')　　B. ('watermelon', 'grape')

C. ('apple', 'grape')　　D. ('orange', 'watermelon')

5（C）：下列哪一个方法可用在元组？(8-7 节)

A. pop( )　　B. insert( )　　C. len( )　　D. append( )

6（D）：下列哪一个方法不可用在元组？(8-7 节)

A. max( )　　B. min( )　　C. len( )　　D. append( )

7（B）：如果想将列表改为元组，可以使用哪一个方法？(8-8 节)

A. list　　B. tuple　　C. append　　D. dict

8（A）：如果想将元组改为列表，可以使用哪一个方法？(8-8 节)

A. list　　B. tuple　　C. append　　D. dict

9（D）：下列哪一个数据类型不可当作 zip( ) 函数的参数？（8-11 节）

A. 元组　　B. 列表　　C. 字典　　D. 整数

第 9 章

一、是非题

1（O）：字典的元素是用 " 键（key）: 值（value）" 配对方式存储。（9-1 节）

2（X）：字典键（key）的值（value）限定是数值（number）或字符串（string）。（9-1 节）

3（X）：有一段程序内容如下：（9-1 节）

```
fruits = {'apple':20, 'orange':25, 'peach':18, 'banana':10}
print(fruits[peach]
```

上述语句可以输出 18。

4（X）：经 clear( ) 删除字典元素后，字典将不再存在于系统。（9-1 节）

5（O）：属于字典 ' 键 ' 的 ' 值 ' 是可以更改的。（9-1 节）

6（O）：字典是无序的数据结构。（9-1 节）

7（O）：Python 允许列表元素是由字典（dict）组成，也允许字典键（key）的值（value）是列表（list）。（9-1 节）

8（X）：可以用键的值（value）判断该元素是否在字典内。（9-1 节）

9（O）：使用 items( ) 方法可以取得字典的键与值。（9-2 节）

10（X）：items( ) 方法所返回字典的键与值是以字典方式存储。（9-2 节）

11（O）：有一个字典 weeks，下列两个程序片段意义相同。（9-2 节）

```
for week in weeks:
    print(week)
```

与

```
for week in weeks.keys( ):
    print(week)
```

12（X）：sorted( ) 方法主要是将字典依值（value）排序。（9-2 节）

13（O）：列表的元素可以是字典。（9-3 节）

14（O）：字典内可以让列表当作元素键的值。（9-4 节）

15（O）：字典内可以让字典当作元素键的值。（9-5 节）

16（X）：fromkeys 是建立字典 " 键 : 值 " 的方法。（9-7 节）

17（X）：将 get( ) 应用在字典时，get( ) 方法的参数是键，如果字典内有找到此键则返回 True。（9-7 节）

二、选择题

1（B）：有一个元组内容是（'ab', 'cd'），可以利用什么函数将此内容转为字典（'a':'b', 'c':'d'）？（9-1 节）

A. update( )　　B. dict( )　　C. len( )　　D. copy( )

2（A）：下列哪一个方法可以合并两个字典为一个字典？（9-1 节）

A. update( )　　B. dict( )　　C. len( )　　D. copy( )

3（A）：下列哪一个方法可以遍历字典的值？（9-2 节）

A. for x in players.values( ):print(x)　　B. for x in players.items( ):print(x)

C. for x in players.keys( ):print(x)　　D. for x in players:print(x)

4（C）：有一个字典内容如下，它的元素数量有几个？（9-1 节）

```
players = {'John':'Golden State', 'age':30, 'height':192}
```

A. 1　　B. 2　　C. 3　　D. 6

5（B）：下列 persons 是一个字典，有一个 for 循环如下：(9-2 节)

```
for info1, info2 in persons.items( ):
    print(info2)
```

上述 info2 可以得到什么？

A. 键　　B. 值　　C. 键 : 值　　D. 字典

6（A）：下列 persons 是一个字典，有一个 for 循环如下：(9-2 节)

```
for info1, info2 in persons.items( ):
    print(info2)
```

上述 info1 可以得到什么？

A. 键　　B. 值　　C. 键 : 值　　D. 字典

7（A）：下列 persons 是一个字典，有一个 for 循环如下：(9-2 节)

```
for info in persons.keys( ):
    print(info)
```

上述 info 可以得到什么？

A. 键　　B. 值　　C. 键 : 值　　D. 字典

8（A）：下列 persons 是一个字典，有一个 for 循环如下：(9-2 节)

```
for info in persons:
    print(info)
```

上述 info 可以得到什么？

A. 键　　B. 值　　C. 键 : 值　　D. 字典

9（B）：下列 persons 是一个字典，有一个 for 循环如下：(9-2 节)

```
for info in persons.values( ):
    print(info)
```

上述 info 可以得到什么？

A. 键　　B. 值　　C. 键 : 值　　D. 字典

10（D）：有一个程序如下：(9-3 节)

```
enemys = []
for enemy_number in range(30):
    enemy = {'color':'red', 'point':5, 'speed':'slow'}
    enemys.append(enemy)
for enemy in enemys[:3]:
    print(enemy)
```

上述程序最后 enemys 字典内有多少个键 : 值元素?

A. 0　　B. 3　　C. 27　　D. 30

11（C）：有一个 Python 数据定义如下：(9-4 节)

```
sports = {'Peter':['NBA', 'NFL'],
          'Mary':['MLB'],
          'John':['NFL', 'MLB', 'NBA']}
```

上述数据定义为何?

A. 列表　　B. 字典内含字典　　C. 字典内含列表　　D. 字典列表

12（B）：有一个 Python 数据定义如下：(9-5 节)

```
persons = {'user1':{
                'name':'Peter',
                'age':25,
                'salary':5000},
           'user2':{
                'name':'Tom',
                'age':30,
                'salary':6500}}
```

上述数据定义为何?

A. 列表　　B. 字典内含字典　　C. 字典内含列表　　D. 字典列表

13（A）：有一程序如下：(9-5 节)

```
persons = {'user1':{
                'name':'Peter',
                'age':25,
                'salary':5000},
           'user2':{
                'name':'Tom',
                'age':30,
                'salary':6500}}
print(len(persons))
```

上述执行结果为何?

A. 2　　B. 3　　C. 4　　D. 8

14（D）：有一个程序片段如下：(9-7 节)

```
x = {1:'a', 2:'b', 3:'c'}
r = x.get(0)
print(r)
```

上述语句可以得到什么结果?

A. a　　B. b　　C. c　　D. None

15（D）：有一个程序片段如下：(9-9 节)

```
wd = 'aabbccd'
dc = {w:wd.count(w) for w in wd}
print(dc)
```

上述语句可以得到什么结果?

A. {'a':1, 'b':1, 'c':1, 'd':1}　　B. {'a':2, 'b':1, 'c':1, 'd':1}

C. {'a':2, 'b':2, 'c':1, 'd':1}　　D. {'a':2, 'b':2, 'c':2, 'd':1}

第 10 章

一、是非题

1（X）：集合是有序的数据，可以用索引取得集合内容。（10-1 节）

2（O）：集合中每一个元素都是唯一的。（10-1 节）

3（X）：集合内有一个元素内容是 'Nelaon'，但发现拼写错误，正确的是 'Nelson'，可以使用 Python 所提供的集合方法将上述元素内容修改正确。（10-1 节）

4（X）：下列指令是定义空集合。（10-1 节）

```
x = {}
```

5（O）：下列指令是定义空集合。（10-1 节）

```
x = set()
```

6（O）：^ 符号是对称差集。（10-2 节）

7（O）：有一个指令片段如下：（10-2 节）

```
wd = 'aabbcc'
for w in set(wd):
    print(w,end='')
```

可能得到 bca 结果。

8（O）：add( ) 方法可以在集合内增加元素，pop( ) 可以随机删除集合的元素。（10-3 节）

9（X）：使用 discard( ) 删除集合元素时，如果元素不存在会导致 KeyError。（10-3 节）

10（X）：集合 A 内容是 {a, b, c}，集合 B 内容是 {d}，有一个指令如下：（10-3 节）

```
boolean = A.issubset(B)
```

上述 boolean 的结果是 True。

11（X）：冻结集合的元素可以增加或减少。（10-5 节）

二、选择题

1（C）：下列哪一个符号可以建立集合？（10-1 节）

A. ( )　　B. [ ]　　C. { }　　D. " "

2（A）：下列哪一种数据类型不可以是集合元素？（10-1 节）

A. 字典　　B. 元组　　C. 整数　　D. 字符串

3（A）：下列哪一种数据类型不可以是集合元素？（10-1 节）

A. 列表　　B. 元组　　C. 整数　　D. 字符串

4（B）：下列哪一种数据类型可以是集合元素？（10-1 节）

A. 字典　　B. 元组　　C. 列表　　D. 集合

5（D）：有一个指令如下：（10-1 节）

```
x = set('aaa bbb cc d')
print(x)
```

上述执行结果是？

A. {'aaabbbccd'}　　B. {'abcd'}

C. {'a', 'b', 'c', 'd'}　　　　D. {'d', ' ', 'c', 'b', 'a'}

6（A）：集合 A 是曾经到美国旅游的人，集合 B 是曾经到英国旅游的人，如果现在想要得到曾经到过两个国家旅游的人，可以使用哪一种集合功能？（6-2 节）

A. 交集　　B. 联集　　C. 差集　　D. 对称差集

7（B）：集合 A 是曾经到美国旅游的人，集合 B 是曾经到英国旅游的人，如果现在想要得到曾经到过美国或英国旅游的人，可以使用哪一种集合功能？（6-2 节）

A. 交集　　B. 联集　　C. 差集　　D. 对称差集

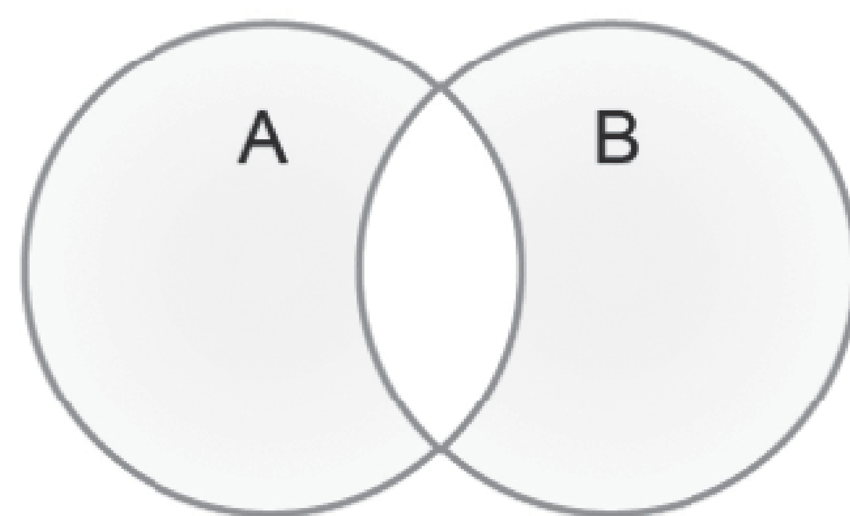
8（D）：集合 A 是曾经到美国旅游的人，集合 B 是曾经到英国旅游的人，如果现在想要得到曾经到英国家但是不曾到过美国旅游的人，可以使用哪一种集合功能？（6-2 节）

A. A & B　　B. A | B　　C. A – B　　D. B - A

9（C）：集合 A 是曾经到美国旅游的人，集合 B 是曾经到英国旅游的人，如果现在想要得到曾经到美国家但是不曾到过英国旅游的人，可以使用哪一种集合功能？（6-2 节）

A. A & B　　B. A | B　　C. A – B　　D. B - A

10（D）：有一个集合 A 和 B 运算图如下：（6-2 节）



如果想在集合运算中取得上述灰色区块，需使用哪一种运算？

A. A & B　　B. A | B　　C. A – B　　D. A ^ B

11（C）：有一指令如下：（10-3 节）

```
A = {'a', 'b', 'c', 'd'}
B = {'a', 'k', 'c'}
C = {'c', 'f', 'w'}
ret_value = A.intersection_update(B)
print(A)
```

上述 A 的结果为何？

A. {'a', 'b', 'c', 'd'}　　　　B. {'b', 'd'}

C. {'a', 'c'}　　　　D. {'a', 'k', 'c'}

12（A）：有一指令如下：（10-3 节）

```
A = {'a', 'b', 'c', 'd'}
B = {'a', 'k', 'c'}
C = {'c', 'f', 'w'}
ret_value = A.intersection_update(B, C)
print(A)
```

上述 A 的结果为何？

A. { 'c', }　　B. {'b', 'd'}　　C. {'a', 'c'}　　D. {'a', 'k', 'c'}

13（B）：有一指令如下：（10-3 节）

```
A = {'a', 'b', 'c', 'd'}
B = {'a', 'k', 'c'}
A.symmetric_difference_update(B)
print(A)
```

上述 A 的结果为何？

A. { 'c', }　　B. {'b', 'd', 'k'}　　C. {'a', 'c'}　　D. {'a', 'k', 'c'}

第 11 章

一、是非题

1（ O ）：程序设计时可能会有一些指令需要重复出现，这时可以思考将重复出现的指令撰写成函数，未来于需要时再加以调用。(11-1 节)

2（ X ）：设计函数时，如果函数参数有默认值，必须将此参数放在参数列的最左边。(11-2 节)

3（ O ）：在调用函数传递参数时，可将参数用 " 参数名称 = 值 " 方式传送，此时若是参数位置错误，程序也可以获得正确结果。(11-2 节)

4（ O ）：设计函数时若是有返回值，可以使用 return 返回。(11-3 节)

5（ X ）：Python 限定函数只能返回一个值。(11-3 节)

6（ O ）：设计函数时如果没有设计 return，Python 直译器也将自动返回 None。(11-3 节)

7（ O ）：有一个函数调用如下：(11-4 节)

```
myfun(a[:], b)
```

上述调用 myfun( ) 所传递的第一个参数 a[:]，其实是传递一个列表副本。

8（ O ）：有一个函数设计如下所示：(11-5 节)

```
myfun(x, *y):
```

上述 *y 代表可以接收 0 到多个参数。

9（ O ）：函数是一种对象，也可以当作列表的元素。(11-6 节)

10（ O ）：函数也可以当作另一个函数的参数。(11-6 节)

11（ O ）：一个函数可以调用自己，这种函数设计称为递归函数。(11-7 节)

12（ O ）：在函数内若是想更改全局变量的值，需在函数内使用 global 声明此全局变量。(11-8 节)

13（ X ）：匿名函数（anonymous function）的名称是 None。(11-9 节)

14（ O ）：高阶函数是指它的部分参数是函数。(11-9 节)

二、选择题

1（ D ）：下列哪一个数据类型不可当作函数的参数？（ 11-2 节 ）

A. 字符串　　B. 元组

C. 函数　　D. 以上皆可当作函数的参数

2（ B ）：设计函数时若没有 return 指令，表示将返回什么？（ 11-3 节 ）

A. 没有返回任何资料　　B. None

C. 函数地址　　D. Error

3（ D ）：某一个函数定义如下：(11-5 节)

```
def fun(*cars):
...
```

调用上述函数时，可以传递多少个参数？

A. 0　　B. 1　　C. 2　　D. 0 到多个

4（B）：有一个函数设计如下所示：(11-5 节)

```
myfun(x, *y):
```

上述 y 的数据类型为何？

A. 列表　　B. 元组　　C. 字典　　D. 集合

5（C）：有一个函数设计如下所示：(11-5 节)

```
myfun(x, **y):
```

上述 y 的数据将是哪一种数据的元素？

A. 列表　　B. 元组　　C. 字典　　D. 集合

6（D）：下列哪一个参数可以接受任意数量的关键词参数？（11-5 节）

A. car　　B. cars　　C. *cars　　D. **cars

7（C）：下列哪一段叙述是错误的？（11-6 节）

A. 函数也是一个对象

B. 函数可以是列表的一个元素

C. 只有函数中的匿名函数可以当作参数传递给其他函数

D. 在函数内可以存在另一个函数

8（C）：下列哪一项不是递归函数的特色？（11-7 节）

A. 函数可以调用自己　　B. 每次函数调用可以使范围越来越少

C. 常用 break 离开函数　　D. 必须要有终止条件

9（D）：下列哪一段叙述是错误的？（11-8 节）

A. 局部变量内容无法在其他函数引用

B. 局部变量内容无法在主程序引用

C. 全局变量内容可以在函数引用

D. 全局变量内容可以随时在函数内更改

10（D）：匿名函数使用下列哪个关键词定义？（11-9 节）

A. def　　B. anonymous　　C. lambda　　D. secret

第 12 章

一、是非题

1（O）：有一个类定义如下：(12-1 节)

```
class A( )
    c = 'silicon stone'
    def d( ):
           pass
```

我们称 c 是属性。

2（O）：有一个类定义如下：(12-1 节)

```
class A( )
    c = 'silicon stone'
    def d( ):
```

```
        pass
```

我们称 d 是方法。

3 (X) : 在类内初始化方法的名称是依程序语意设置的。(12-1 节)

4 (O) : 面向对象程序语言的基本精神是类的属性经过封装 (encapsulation) 后，类外无法直接更改其内容。(12-2 节)

5 (O) : 在 Python 类中私有属性是名称前面增加 __ (两个下画线)。(12-2 节)

6 (X) : 基类也可称为子类。(12-3 节)

7 (X) : 衍生类也可称为父类。(12-3 节)

8 (X) : 有一个基类 X，此类有两个子类 A 和 B，A 类无法取得 B 类的属性。(12-3 节)

9 (X) : 面向对象的多态 (polymorphism) 限定彼此是衍生类的关系。(12-4 节)

10 (O) : Python 允许一个类有多个衍生类，也允许多个基类有一个衍生类。(12-5 节)

11 (O) : 在类应用中，可以使用 type( ) 获得某一个类对象的类名称。(12-6 节)

12 (O) : Python 的 isinstance( ) 可以判断一个对象是不是属于特定类。(12-6 节)

13 (X) : 在 Python 的程序执行中 __name__ 一定是 __main__。(12-7 节)

二、选择题

1 (D) : 以下哪一个是使用 Python 时，自建的数据类型？ (12-1 节)

A. 集合 (set)　　B. 列表 (list)　　C. 字典 (dict)　　D. 类 (class)

2 (B) : 以下哪一个是类初始化方法的名称？ (12-1 节)

A. init　　B. __init__　　C. main　　D. __main__

3 (A) : 以下哪一个是初始化方法的第一个参数？ (12-1 节)

A. self　　B. init　　C. constructor　　D. begin

4 (C) : 以下哪一个是私有属性名称前面的字符串？ (12-2 节)

A. private　　B. **　　C. __　　D. --

5 (C) : 衍生类引用基类的初始化方法要用哪一个方法？ (12-3 节)

A. __init__( )　　B. __iter__( )　　C. super( )　　D. __getitem__( )

6 (A) : 有一个程序如下 : (12-6 节)

```
class A():
    def fn(self):
        pass

a = A()
print(type(a.fn))
```

输出结果为何？

A. <class 'method'>　　B. <class '__main__.A'>

C. <class 'function'>　　D. <class 'str'>

7 (B) : 有一个程序如下 : (12-6 节)

```
class A():
    def fn(self):
        pass

a = A()
print(type(a))
```

输出结果为何？

A. <class 'method'>　　B. <class '__main__.A'>

C. <class 'function'>　　D. <class 'str’ >

8（A）：有一个程序如下：(12-6 节）

```
class A():
    pass

class B(A):
    pass

class C(B):
    pass

d = B()
print(isinstance(d, C))
```

输出结果为何？

A. True　　B. False　　C. d, C　　D. B

9（B）：有一个程序如下：(12-6 节）

```
class A():
    pass

class B(A):
    pass

class C(B):
    pass

d = B()
print(isinstance(d, A))
```

输出结果为何？

A. True　　B. False　　C. d, A　　D. A

10（B）：如果执行独立的 a.py 程序，打印 __name__ 可以得到？(12-7 节）

A. __name__　　B. __main__　　C. __a__　　D. __doc__

第 13 章

一、是非题

1（X）：Python 模块的扩展名是 mod。(13-1 节）

2（X）：Python 可由程序的扩展名判断这是一般程序或模块程序。(13-1 节）

3（O）：使用 "import 模块名称 " 导入模块时，如果要引用 cooking( ) 函数，语法格式如下：(13-2 节）

模块名称 .cooking( )

4（O）：假设有一个 Python 程序片段如下：(13-2 节）

```
from car import battery
```

从上述可知，模块名称是 car。

5（X）：假设有一个 Python 程序片段如下：(13-2 节）

```
from car import battery
```

从上述可知，导入模块的函数是 car。

6（O）：Python 允许给导入的模块函数替代名称，也允许给模块替代名称。（13-2 节）

7（X）：一个模块只能放一个类。（13-3 节）

8（O）：程序设计师可以使用随机数的概念控制网络游戏庄家和玩家的输赢比例。（13-5 节）

9（X）：randint(1, 10) 可以产生大于等于 0 和小于 10 的随机数。（13-5 节）

10（O）：random 模块的 random( ) 可以产生随机的浮点数。（13-5 节）

11（X）：sys.time( ) 方法可以返回自 2000 年 1 月 1 日 00:00:00AM 以来的秒数。（13-6 节）

12（O）：sys.executable 可以获得目前设计 Python 程序的文件路径。（13-7 节）

13（O）：sys.stdout 是一个对象，主要是用于 Python Shell 窗口的输出。（13-7 节）

14（O）：calendar 模块的 calendar( ) 可以打印出年历。（13-9 节）

二、选择题

1（B）：在 Python 中使用下列语法导入多个函数时，各函数间可以用什么符号区隔？（13-2 节）

```
from module_name import functions
```

假设上述 functions 是一系列函数。

A. 句号 "."　　B. 逗号 ","　　C. 分号 ";"　　D. 等号 "="

2（D）：Python 语言中在 "from 模块名称 import　xx" 右边 xx 是什么符号代表导入所有函数？（13-2 节）

A. 句号 "."　　B. 逗号 ","　　C. 分号 ";"　　D. "*"

3（A）：有一个语法如下：（13-2 节）

```
import module_name xx alternative_name
```

上述 xx 可能是什么关键词？

A. as　　B. for　　C. while　　D. raise

4（B）：下列哪一个方法可以重组列表的顺序？（13-5 节）

A. sample( )　　B. shuffle( )　　C. choice( )　　D. time( )

5（C）：下列哪一个方法可以随机返回列表的元素？（13-5 节）

A. sample( )　　B. shuffle( )　　C. choice( )　　D. time( )

6（A）：下列哪一个方法可以随机返回第 2 个参数数量的列表元素？（13-5 节）

A. sample( )　　B. shuffle( )　　C. choice( )　　D. time( )

7（D）：下列哪一个方法返回的数据无法判断目前系统时间？（13-6 节）

A. time( )　　B. asctime( )　　C. localtime( )　　D. sleep( )

8（B）：下列哪一个方法返回的数据为可清楚阅读的系统时间？（13-6 节）

A. time( )　　B. asctime( )　　C. localtime( )　　D. sleep( )

9（C）：下列哪一个方法返回的数据可用索引 [7] 得到目前系统日期是今年的第几天？（13-6 节）

A. time( )　　B. asctime( )　　C. localtime( )　　D. sleep( )

10（C）：哪一个模块提供 version 属性，可以得到目前 Python 系统版本信息？（13-7 节）

A. time　　B. keyword　　C. sys　　D. random

11（A）：哪一个可以返回目前 Python 的使用平台？（13-7 节）

A. platform B. path C. executable D. version

12（B）：下列哪一个 keyword 模块的属性是 Python 关键词？（13-8 节）

A. iskey B. kwlist C. name D. path

第 14 章

一、是非题

1（X）：在相对路径概念中 ".." 代表根目录。（14-1 节）

2（O）：在相对路径概念中 "." 代表目前工作目录。（14-1 节）

3（O）：使用 Python 可以获得特定文件的大小信息。（14-1 节）

4（O）：使用 with 配合 open( ) 打开文件时，会在不需要此文件时自动关闭文件。（14-2 节）

5（X）：使用 readlines( ) 读取文件时，是一次读取一行，然后用字典（dict）方式存储。（14-2 节）

6（O）：使用 find( ) 查找字符串时，如果没有找到会返回 -1。（14-2 节）

7（X）：使用 rfind( ) 查找字符串时，会返回查找字符串第一次出现的位置。（14-2 节）

8（O）：使用 write( ) 时如果是输出数值数据会产生错误。（14-3 节）

9（O）：open( ) 方法也可以打开二进制文件，未来可以用 read( ) 读取此二进制文件的内容。（14-4 节）

10（X）：读取二进制文件时，必须读取每个字节，才可以读到文件的最后位置。（14-4 节）

11（X）：使用 shutil 模块可以处理文件的复制与移动，但是只限于在目前工作目录下进行。（14-5 节）

12（O）：shutil 模块的 move( ) 方法除了可以执行文件移动与更改，也可以执行目录移动与更改。（14-5 节）

13（O）：有一个文件内含下列指令。（14-6 节）

```
x = zipfile.ZipFile('xout.zip', 'w')
```

上述 xout.zip 是未来储存压缩文件的文件名，x 是压缩文件的文件对象，未来可以调用 write( ) 方法将压缩结果存入 xout.zip。

14（X）：在中文 Windows 操作系统环境，Python 的 open( ) 默认打开文件的编码格式是 utf-8。（14-7 节）

15（O）：BOM（Byte Order Mark）俗称文件前端代码，主要功能是判断文字以 Unicode 表示时，字节的排列方式。（14-7 节）

二、选择题

1（A）：下列哪一个方法可以获得目前工作目录？（14-1 节）

A. getcwd( ) B. walk( ) C. mkdir( ) D. chdir( )

2（C）：下列哪一个模块可以使用通配符 *，列出特定工作目录文件信息？（14-1 节）

A. os B. os.path C. glob D. zipfile

3（B）：下列哪一个方法可以遍历目录树？（14-1 节）

A. getcwd( ) B. walk( ) C. mkdir( ) D. chdir( )

4（A）：open( ) 方法默认 mode=?，请问 ? 是什么？（14-2 节）

A. 'r'　　B. 'w'　　C. 'a'　　D. 'c'

5（C）：open( ) 在哪一关键词内使用，未来不需要时可以不必使用 close( )？（14-2 节）

A. raise　　B. assert　　C. with　　D. break

6（C）：如果打开文件是要将文件输出到文件的末端，open( ) 内需要加上哪一个参数？（14-3 节）

A. 'r'　　B. 'w'　　C. 'a'　　D. 'c'

7（D）：下列哪一个 mode 参数是打开供写入？（14-4 节）

A. 'r'　　B. 'w'　　C. 'rb'　　D. 'wb'

8（B）：下列哪一个方法可以返回目前读取二进制文件时，读写指针从文件开头算起的指针位置？（14-4 节）

A. seek( )　　B. tell( )　　C. origin( )　　D. walk( )

9（A）：下列哪一个方法可以移动读取二进制文件时，读写指针位置？（14-4 节）

A. seek( )　　B. tell( )　　C. origin( )　　D. walk( )

10（D）：shutil 模块的 move( ) 无法执行下列哪一个工作？（14-5 节）

A. 文件名的更改　　B. 目录名称的更改

C. 文件或目录的移动　　D. 目录的删除

11（A）：下列哪一个方法可以执行目录名称的更改？（14-5 节）

A. move( )　　B. rmtree( )　　C. send2trash( )　　D. copytree( )

12（B）：下列哪一个方法删除文件或目录后可以在资源回收站找回？（14-5 节）

A. del( )　　B. send2trash( )　　C. rmdir( )　　D. rmtree( )

13（C）：下列哪一个模块可以执行压缩或解压缩 zip 文件？（14-6 节）

A. zipfile　　B. os　　C. shutil　　D. send2trash

14（C）：下列哪一个是多语系的编码规则，使用可变长度字节方式存储字符？（14-7 节）

A. cp-950　　B. gb2312　　C. utf-8　　D. ANSI

15（D）：下列哪一个模块的 copy( ) 方法可以将字符串数据复制至剪贴板？（14-8 节）

A. zipfile　　B. shutil　　C. os　　D. pyperclip

第 15 章

一、是非题

1（X）：在 try – except 指令中，如果 try 下面的指令是有错误的，一定会执行 except 的错误处理程序。（15-1 节）

2（O）：在 try – except 指令中，如果 try 下面的指令是正常的，一定会跳开 except 的错误处理程序。（15-1 节）

3（O）：在 try – except 的使用中，可以使用多个 except 捕捉多个异常。（15-2 节）

4（O）：在 try – except 的使用中，可以使用一个 except 捕捉多个异常。（15-2 节）

5（X）：使用 Python 设计程序时，异常的判定由直译器判定，无法自行建立异常的标准。（15-3 节）

6（O）：traceback 模块内有 traceback.format_exc( ) 方法，可以记录 Traceback 字符串。（15-4 节）

7（O）：真实计算机上的第一只虫是蛾（moth）。（15-6 节）

二、选择题

1（C）：Python 程序错误消息的标注字符串是什么？（15-1 节）

A. Error　　B. Message　　C. Traceback　　D. Warning

2（A）：抛出除数为 0 的异常消息是什么？（15-1 节）

A. ZeroDivisionError　　B. FileNotFoundError

C. TypeError　　D. ValueError

3（B）：找不到所打开文件的异常消息是什么？（15-1 节）

A. ZeroDivisionError　　B. FileNotFoundError

C. TypeError　　D. ValueError

4（C）：以字符当作除数或被除数运算时，所产生的异常是什么？（15-1 节）

A. ZeroDivisionError　　B. FileNotFoundError

C. TypeError　　D. ValueError

5（D）：在 try – except 的使用中，哪一个可捕捉一般的异常？（15-2 节）

A. ZeroDivisionError　　B. FileNotFoundError

C. TypeError　　D. Exception

6（D）：使用 Python 程序设计时，我们自行定义异常时同时丢出异常的关键词？（15-3 节）

A. except　　B. try　　C. finally　　D. raise

7（A）：在 write( ) 内放哪一个参数可以将 Traceback 字符串写入文件？（15-4 节）

A. traceback.format_exc( )　　B. raise.exc( )

C. data.format_exc( )　　D. file.exc( )

8（C）：下列哪一个关键词需要与 try 配合使用，同时不论是否有异常发生一定会执行这个关键词内的程序代码？（15-5 节）

A. except　　B. else　　C. finally　　D. raise

第 16 章

一、是非题

1（X）：Python 使用正则表达式时，re.compile( ) 是必需的，将正则表达式放在方法内当参数，这个程序不可省略。（16-2 节）

2（X）：re.search( ) 查找失败时，会返回空字符串。（16-2 节）

3（O）：re.search( ) 查找时，如果成功只返回第一个查找到的字符串。（16-2 节）

4（X）：re.findall( ) 查找时，如果成功只返回第一个查找到的字符串。（16-2 节）

5（O）：re.findall( ) 查找失败时，会返回空列表。（16-2 节）

6（X）：使用 re.search( ) 时，如果正则表达式有分组，group（0）可以返回比对括号的第一组文字。（16-3 节）

7（O）：当我们使用 re.search( ) 查找字符串时，可以使用 groups( ) 方法取得分组的内容。（16-3 节）

8（X）：管道在逻辑概念中，可想成是 AND 的概念。（16-3 节）

9（ X ）：Python 默认的查找模式是非贪婪模式。(16-4 节)

10（ X ）：有一个 pattern = '^Mary'，msg = "She is Mary'，执行下列指令后

```
txt = re.findall(pattern, msg)
```

最后 txt 的内容是 ['Mary'] 。(16-5 节)

11（ O ）："." 是通配符，但是只限定一个字符，同时不可当作换行字符。(16-5 节)

12（ O ）：re.DOTALL 参数允许查找时碰上换行字符将继续执行。(16-5 节)

13（ O ）：re.match( ) 重要概念是如果开始字符比对失败，整个查找就算失败。(16-6 节)

14（ O ）：span( ) 可想成是 start( ) 和 end( ) 的组合。(16-6 节)

15（ O ）：sub( ) 除了可以执行字符串替代，也可以用隐藏方式执行字符串替代。例如，用 *** 替代一些符合比对的字符串。(16-7 节)

二、选择题

1（ C ）：有一个正则表达式是 "r'\d{3}"，下列哪一个字符串符合规定？（ 16-2 节)

A. a12　B. 13a　C. 123　D. abc

2（ D ）：如果所查找的正则表达式字符串有用小括号分组时，若是使用 findall( ) 方法处理，会返回列表，列表内的元素是哪一种数据类型？（ 16-3 节)

A. 字符串　B. 列表　C. 字典　D. 元组

3（ A ）：下列哪一个符号在正则表达式的查找比对时，代表前方括号的正则表达式或字符串是可有可无？（ 16-3 节)

A. ?　B. +　C. .　D. *

4（ D ）：下列哪一个符号在正则表达式的查找比对时，代表前方括号的正则表达式或字符串是可从 0 到多次？（ 16-3 节)

A. ?　B. +　C. .　D. *

5（ B ）：下列哪一个符号在正则表达式的查找比对时，代表前方括号的正则表达式或字符串是可从 1 到多次 ? (16-3 节)

A. ?　B. +　C. .　D. *

6（ B ）：下列哪一个参数可以让正则表达式的查找比对时忽略大小写？（ 16-3 节)

A. re.NONECASE　B. re.IGNORECASE

C. re.DOTALL　D. re.VERBOSE

7（ A ）：哪一个符号可以将正则表达式的查找由贪婪模式改成非贪婪模式？（ 16-4 节)

A. ?　B. +　C. .　D. *

8（ B ）：在正则表达式中，哪一个是代表 0 ～ 9 的数字？（ 16-5 节)

A. \s　B. \d　C. \w　D. \k

9（ B ）：空格符是哪一种正则表达式的字符？（ 16-5 节)

A. \d　B. \s　C. \w　D. \b

10（ D ）：有一个正则表达式是 [^aeiouAEIOU]，下列哪一个字符符合查找条件？（ 16-5 节)

A. a　B. O　C. u　D. z

11（ D ）：下列哪一个参数可以让正则表达式内部有注释文字 ? (16-8 节)

A. re.NONECASE　B. re.IGNORECASE

C. re.DOTALL　D. re.VERBOSE

第 17 章

一、是非题

1（ X ）：在 Pillow 模块的 RGBA 概念中，A 是 Alpha，此值越大代表透明度越高。(17-1 节)

2（ O ）：使用 Pillow 模块可以将 jpg 文件转存成 png 文件格式。(17-3 节)

3（ X ）：Pillow 模块的 RGBA 模式一般建议是建立 jpg 文件格式的图像对象。(17-3 节)

4（ O ）：Pillow 模块允许为图像的一个像素编辑。(17-4 节)

5（ O ）：Pillow 模块允许在某一张图片内，插入另一张图片，达到图像合成的效果。(17-5 节)

6（ O ）：Pillow 模块内有 ImageFilter 模块，这个模块内有功能可以为图片加上滤镜效果。(17-6 节)

7（ O ）：Pillow 模块内有 ImageDraw 模块，这个模块内有功能可以在图片内绘制图形或是书写文字。(17-7 和 17-8 节)

8（ X ）：ImageDraw 模块允许在图像内填写英文，但是不支持填写中文。(17-8 节)

9（ X ）：词云（Word Cloud）是 Python 内建的模块。(17-10 节)

10（ O ）：建立矩形的词云（Word Cloud）时，也可以自行设置词云图像文件的大小。(17-10 节)

11（ O ）：有许多方法可以产生词云（Word Cloud）的图像文件。(17-10 节)

二、选择题

1（ C ）：在 Pillow 模块中，由 size 属性可以获得图像的哪些信息？（ 17-3 节 ）

A. 只有宽度　　B. 只有高度　　C. 宽度和高度　　D. 面积

2（ A ）：在 Pillow 模块中，图像对象的哪一个属性可以返回图像对象文件的扩展名？(17-3 节)

A. format　　B. filename　　C. size　　D. save

3（ C ）：下列哪一个方法可以让图像翻转？（ 17-4 节 ）

A. rotate( )　　B. copy( )　　C. transpose( )　　D. paste( )

4（ D ）：下列哪一个方法可以合成图像？（ 17-5 节 ）

A. rotate( )　　B. copy( )　　C. transpose( )　　D. paste( )

5（ B ）：在 Pillow 模块的 ImageFilter 模块中下列图片是哪一种滤镜效果？（ 17-6 节 ）



A. BLUR　　B. CONTOUR　　C. EMBOSS　　D. FIND_EDGES

6（ A ）：Pillow 模块内的哪一个模块支持在图像内填写文字？(17-8 节)

A. ImageDraw　　B. ImageFilter　　C. ImageColor　　D. ImageWord

7（D）：下列哪一个模块支持建立 QR code 信息？（17-9 节）

A. ImageDraw　　B. Pillow　　C. ImageFont　　D. qrcode

8（B）：下列哪一个模块可以建立中文分词？（17-10 节）

A. wordcloud　　B. jieba　　C. matplotlib　　D. PIL

9（C）：建立非矩形外观的词云，需增加下列哪一个参数？（17-10 节）

A. font_path　　B. background_color　　C. mask　　D. width

第 18 章

一、是非题

1（X）：使用 tkinter 所设计的 GUI 程序限定只能在 Windows 操作系统下执行。(18-1 节)

2（O）：在 Python 程序中使用 resizeable(0,0) 表示无法更改窗口的宽度与高度。(18-1 节)

3（X）：tkinter 的标签 Label 目前只可以有文字功能。(18-2 节)

4（X）：使用 pack( ) 方法做窗口组件定位时默认是从左到右排列。(18-3 节)

5（O）：使用 tkinter 在窗口建立功能按钮时，可以将文字或是图像应用在功能按钮上。(18-4 节)

6（X）：tkinter 的文字区域 Text 组件限定只能在此输入 1 行数据。(18-7 节)

7（X）：Radiobutton 是用在系列选项中可以有多个选择。(18-9 节)

8（O）：Checkbutton 是用在系列选项中可以有多个选择。(18-10 节)

二、选择题

1（B）：下列哪一个方法是用 row 和 column 参数执行窗口组件的定位？（18-3 节）

A. pack( )　　B. grid( )　　C. place( )　　D. set( )

2（C）：下列哪一个方法是用 x,y 参数定义窗口组件的位置？（18-3 节）

A. pack( )　　B. grid( )　　C. place( )　　D. set( )

3（C）：下列哪一个方法可以执行窗口组件配置，但是却不鼓励读者使用？（18-3 节）

A. pack( )　　B. grid( )　　C. place( )　　D. set( )

4（B）：下列哪一个方法不是 tkinter 支持的变量类型？（18-5 节）

A. Intvar( )　　B. FloatVar( )　　C. StringVar( )　　D. BeeleanVar( )

5（C）：如果想要建立一个文本编辑器，下列哪一个窗口组件是必要的？（18-6 节）

A. Label　　B. Button　　C. Text　　D. Radiobutton

6（A）：下列哪一个 tkinter 窗口组件适合设计从 5 个兴趣项目中勾选 1 个自己最感兴趣的事物？（18-9 节）

A. Radiobutton　　B. CheckButton　　C. Label　　D. Scale

7（B）：下列哪一个 tkinter 窗口组件适合设计从 5 个兴趣项目中勾选多个自己感兴趣的事物？（18-10 节）

A. Radiobutton　　B. CheckButton　　C. Label　　D. Scale

8（D）：下列哪一个组件不可以当作数值输入？（18-14 节）

A. Scale　　B. Entry　　C. Text　　D. Menu

第 19 章

一、是非题

1（O）：Canvas 是属于 tkinter 模块内的一个组件（widget）。(19-1 节)

2（O）：create_line( ) 可以绘制不同颜色与粗细的线条。(19-1 节)

3（O）：create_oval( ) 方法除了可以绘制圆也可以绘制椭圆。(19-1 节)

4（O）：在赌场用计算机控制的赛马程序，其实庄家可以控制本身的输赢。(19-3 节)

5（X）：设计动画游戏时，物体每次移动的距离与物体的移动速度无关。(19-3 节)

6（X）：反弹球在上下垂直移动时，每次需调整的是球所在的 x 轴位置。(19-3 节)

二、选择题

1（D）：下列哪一个方法与设置画布背景颜色有关？（19-2 节）

A. move( )　　B. update( )　　C. sleep( )　　D. Scale( )

2（D）：下列哪一个方法不是绘制动画的必要方法？（19-3 节）

A. move( )　　B. update( )　　C. sleep( )　　D. Scale( )

3（B）：下列哪一个方法可以重绘画布？（19-3 节）

A. move( )　　B. update( )　　C. sleep( )　　D. Scale( )

第 20 章

一、是非题

1（X）：使用 plot(data) 方法绘制图表时，此方法内有一个列表 data，列表 data 内的值会被视为 x 轴的值。(20-1 节)

2（O）：使用 plot(a,b) 方法绘制图表时，此方法内有两个列表 a 和 b，a 列表代表 x 轴的值，b 列表代表 y 轴的值。(20-1 节)

3（X）：使用 plot(a,b,c,d) 方法绘制图表时，此方法内有 4 个列表 a、b、c 和 d，这是绘制立体图，c 代表 z 轴的值，d 代表时间轴。(20-1 节)

4（O）：使用 matplotlib 模块时，savfig( ) 可以存储图表，show( ) 可以显示图表。(20-1 节)

5（O）：可以使用 scatter( ) 方法绘制三角函数的 sin 或 cos 的波形图。(20-2 节)

6（O）：bar( ) 方法可以建立直方图。(20-6 节)

7（O）：pie( ) 方法可以建立圆饼图。(20-7 节)

8（X）：matplotlib 模块默认是图表可以显示中文。(20-8 节)

二、选择题

1（D）：下列哪一个方法可以设置坐标轴的刻度？（20-1 节）

A. title　　B. xlabel　　C. ylabel　　D. tick_params

2（A）：使用 matplotlib 模块时，下列哪一个方法可以建立图表标题？（20-1 节）

A. title　　B. legend　　C. xlabel　　D. show

3（B）：arrange( ) 可以产生数组，请问这是哪一个模块的方法？（20-3 节）

A. matplotlib　　B. Numpy　　C. tkinter　　D. turtle

4（C）：有一个 subplot（a,b,c）方法，c 方法所代表的意义是什么？（20-5 节）

A. 垂直方向要绘制几张图　B. 水平方向要绘制几张图
C. 这是第几张图　D. 总共有几张图

5（B）：下列哪一个参数可以设置圆饼图区块可以分离？（20-7 节）

A. labels　B. explode　C. autopct　D. labeldistance

第 21 章

一、是非题

1（O）：JSON 对象是用大括号表示，内部元素是用键 : 值方式配对存储。（21-1 节）

2（O）：JSON 的 dumps( ) 方法的 indent 参数可以设置缩排，让对象更容易显示。（21-2 节）

3（X）：使用 dumps( ) 将 Python 资料转成 JSON 格式时，如果 Python 数据是 dict，可以转成 string。（21-2 节）

二、选择题

1（A）：下列哪一项是正确的 JSON 对象？（21-1 节）

A. {"Name":"Kevin", "Age":25}　B. {"Name":"Tomy", 25:32}
C. ["Name":"Kevin", "Age":25]　D. ["Name", "Kevin", "Age", 25

2（B）：json 模块的 dumps( ) 可以将 Python 数据的 tuple 转成哪一种字符串格式？（21-2 节）

A. object　B. array　C. string　D. number

3（C）：下列哪一项可以将 JSON 格式数据转成 Python 数据？（21-2 节）

A. dumps( )　B. dump( )　C. loads( )　D. load( )

4（B）：下列哪一项可以将 Python 数据转成 JSON 文件？（21-3 节）

A. dumps( )　B. dump( )　C. loads( )　D. load( )

5（D）：下列哪一项可以读取 JSON 文件？（21-3 节）

A. dumps( )　B. dump( )　C. loads( )　D. load( )

第 22 章

一、是非题

1（O）：CSV 文件可以用 Windows 内附的记事本打开，也可以用 Microsoft Excel 打开。（22-1 节）

2（O）：'\t' 也可以当作 CSV 文件的分隔符。（22-5 节）

二、选择题

1（C）：CSV 文件使用记事本打开时，数据字段主要分隔字符是（　　）。（22-1 节）

A. =　B. @　C. ,　D. ^

2（B）：可以将列表资料输出至 CSV 文件。（22-5 节）

A. writer( )　B. writerow( )　C. output( )　D. print( )

3（A）：可以让图表轴坐标的标记旋转。（22-6 节）

A. autofmt_xdate( )　B. axis( )
C. rotate( )　D. tick_params( )

第 23 章

一、是非题

1 (O)：Numpy 模块处理数组数据，处理速度比 Python 的列表（list）快。(23-1 节)

2 (X)：Numpy 模块的 itemsize 属性代表数组元素个数。(23-3 节)

3 (X)：Numpy 模块的 zeros( ) 默认是建立 0 的整数数组。(23-3 节)

4 (X)：向量内积其实就是数组的乘法。(23-3 节)

5 (O)：reshape( ) 可以更改数组外形。(23-4 节)

6 (O)：ravel( ) 可以将多维数组转成一维数组。(23-4 节)

7 (O)：Numpy 模块的广播（broadcast）机制是将比较小的数组扩大至与较大的数组外形相同，然后再执行运算。(23-5 节)

8 (X)：Numpy 模块的 floor( ) 可以返回大于或等于的最小整数。(23-7 节)

9 (O)：permutation( ) 返回数据是数组。(23-8 节)

10 (X)：mean( ) 主要功能是如果有 weights 则返回数组的加权平均。(23-9 节)

二、选择题

1 (D)：下列哪一个不是 Numpy 数组数据类型 ndarry 的特点？ (23-1 节)

A. 数组大小固定　　B. 数组内容数据类型相同

C. 支持复数运算　　D. 执行速度比较慢

2 (B)：下列哪一个是数组维度属性？ (23-3 节)

A. itemsize　　B. ndim　　C. shape　　D. size

3 (D)：有一个运算如下：(23-3 节)

```
>>> x = np.array([1,2,3,4,5])
>>> y = np.insert(x, 2, 8)
>>> print(y)
```

可以得到下列哪个结果？

A. [1,2,3,4,5,8]　　B. [8,1,2,3,4,5]　　C. [1,8,2,3,4,5]　　D. [1,2,8,3,4,5]

4 (B)：有一个运算如下：(23-3 节)

```
>>> x = np.array([1,2,3,4,5])
>>> y = np.insert(x, [1,3], [7,9])
>>> print(y)
```

可以得到下列哪个结果？

A. [1,2,3,4,5,7,9]　　B. [1,7,2,3,9,4,5]　　C. [1,2,7,3,4,5,9]　　D. [7,1,2,9,3,4,5]

5 (C)：有一个运算如下：(23-4 节)

```
>>> x = np.arange(6).reshape(2,3)
>>> print(x[0])
```

可以得到下列哪个结果？

A. 0　　B. [0]　　C. [0 1 2]　　D. [1 2 3]

6 (A)：有一个运算如下：(23-4 节)

```
>>> x = np.array([[1,2,3,4],[2,3,4,5],[3,4,5,6]])
>>> print(x[0:3,0])
```

可以得到下列哪个结果？

A. [1 2 3]　　B. [2 3 4]　　C. [3 4 5]　　D. [4 5 6]

7（C）：下列哪一个函数可以返回小于或等于的最大整数？（23-7 节）

A. rint( )　　B. around( )　　C. floor( )　　D. ceil( )

8（B）：有一个运算如下：(23-7 节)

```
>>> x = np.maximum([1, 5, 10],[2, 6, 9])
>>> print(x)
```

可以得到下列哪个结果？

A. [1 5 9]　　B. [2 6 10]　　C. [2 5 10]　　D. [1 5 10]

9（B）：有一个运算如下：(23-9 节)

```
>>> x = np.array([[12,7,4],[3,2,6]])
>>> np.median(x)
```

可以得到下列哪个结果？

A. 4.0　　B. 5.0　　C. 6.0　　D. 7.0

第 24 章

一、是非题

1（O）：质量概率函数（probability mass function）是指离散随机数在特定值上的概率。(24-2 节)

2（X）：使用 scipy.optimize 模块的 root( ) 方法，可以使用一个初始迭代值即可求出一元二次方程式的所有根。(24-3 节)

3（X）：下列函数可以使用 minimize.scalar( ) 函数找出最小值。(24-3 节)

$$f(x) = -3(x-2)^2 + 3$$

4（O）：插值是由一些已知的数据散点，使用插入方法推估新的点。(24-4 节)

二、选择题

1（D）：线性代数模块 scipy.linalg 无法处理下列哪一个数学问题？（24-1 节）

A. 计算行列式　　B. 解联立方程式　　C. 求特征值　　D. 以上皆可

2（A）：ppf( ) 函数是哪一个函数的逆函数？（24-2 节）

A. cdf( )　　B. pmf( )　　C. rvs( )　　D. pdf( )

3（C）：在二项分布实验中，假设成功概率是 p，如果实验 n 次则失败次数是多少？(24-2 节)

A. 1-p　　B. np　　C. n(1-p)　　D. n(1+p)

第 25 章

一、是非题

1（O）：Pandas 是一维数组结构。(25-1 节)

2（O）：使用字典（dict）建立 Series 时，字典的键（key）会被视为 Series 对象的索引。(25-1 节)

3（X）：Series 的索引一定是从 0 开始，无法更改。（25-1 节）

4（X）：DataFrame 是一维的数组结构。（25-2 节）

5（O）：concat( ) 可以将两个 Series 组合成 DataFrame。（25-2 节）

6（X）：如果使用 concat( ) 将多个 Series 组成 DateFrame，参数需设置 axis=0。（25-2 节）

7（O）：使用元素是字典的列表建立 DataFrame，字典的键（key）将是 DataFrame 的域名。（25-2 节）

8（O）：任何数值与 NaN 做运算时，结果都是 NaN。（25-3 节）

9（X）：使用 Pandas 的 sort_values( ) 排序时，默认是从大排到小。（25-3 节）

10（O）：CSV 文件可以用 Microsoft Excel 打开。（25-4 节）

11（X）：to_csv( ) 可以读取 csv 文件。（25-4 节）

12（X）：一个图表只能有 x 和 y 轴。（25-5 节）

二、选择题

1（B）：Pandas 的 Series 架构与 Python 的哪一个数据结构类似？（25-1 节）

A. 字典　B. 列表　C. 元组　D. 集合

2（C）：有一个程序如下，最后输出结果为何？（25-1 节）

```
>>> s = pd.Series([11, 33, 55, 77, 99])
>>> print(s[3])
```

A. 11　B. 33　C. 55　D. 77

3（D）：有一个程序如下，最后输出结果为何？请忽略索引值。（25-1 节）

```
>>> s = pd.Series(np.arange(0, 10, 2))
>>> print(s[-1:])
```

A. 0　B. 2　C. 6　D. 8

4（D）：有一个程序如下，最后输出会有几个 NaN ？请忽略索引值。（25-1 节）

```
>>> index1 = [1, 3, 5, 7, 9]
>>> index2 = [2, 4, 6, 8, 10]
>>> s1 = pd.Series([10, 20, 30, 40, 50], index=index1)
>>> s2 = pd.Series([10, 20, 30, 40, 50], index=index2)
>>> y = s1 + s2
>>> print(y)
```

A. 0　B. 5　C. 8　D. 10

5（B）：要想用 concat( ) 方法将多个 Series 对象组成 DataFrame，其中参数 axis 需设为什么？（25-1 节）

A. 0　B. 1　C. 2　D. 3

6（C）：下列哪一个方法可以使用 index 或 columns 内容取得或设置 DataFrame 整个 row 或 columns 数据或数组内容？（25-2 节）

A. at　B. iat　C. loc　D. iloc

7（A）：下列哪一个方法可以将 NaN 删除？（25-3 节）

A. dropna( )　B. fillna( )　C. isna( )　D. notna( )

8（A）：CSV 文件各字段间的默认分隔字符是什么？（25-4 节）

A. ,　B. ?　C. #　D. @

9（D）：下列哪一个模块与网络爬虫有关？（25-7 节）

A. matplotlib　B. numpy　C. pandas　D. requests